

Greg,

Here is an updated version of the sysfs tagged directories that improves a bit the situation over the previous one.

I've modified the patch 09 ("Enable tagging for net\_class directories in sysfs") to be a bit less intrusive in sysfs core. I removed the #ifdef'd parts you didn't like in fs/sysfs/mount.c, and replaced it by a generic routine `sysfs_ns_exit()` that is called, if needed, by the namespace when it exits. This routine goes through every sysfs super blocks and calls the callback passed by the namespace to clean its tag.

The patch is now splitted in two:

- \* 09/11: the generic routine,
- \* 10/11: the remaining network parts.

The generic part can be merge with patch 05 ("sysfs: Implement sysfs tagged directory support.") but I left it separate for now to ease reviews.

Serge's patch for user namespace is modified to use this new service too. No more #ifdef CONFIG\_NET or #ifdef CONFIG\_USER\_NS in fs/sysfs/mount.c now.

But, currently, a new namespace that wants to add its tag to sysfs dirs still need to modify fs/sysfs/mount.c in a few routines to manage the new tag member added in struct `sysfs_tag_info`: `sysfs_fill_super()`, `sysfs_test_super()`, `sysfs_kill_sb()` (see the last two patches). These changes are only the initialization and a bunch of comparisons.

If we really want to go further, to get rid of these, I've thought about:

- \* Extending `sysfs_tagged_dir_operations` with super blocks operations:
  - `fill_sb_tag`, `test_sb_tag`, `kill_sb_tag`
- \* Add routines in sysfs to allow registration/unregistration of these operations structs in a list:
  - `sysfs_register_tagged_dir_ops()`...
- \* Each subsystem concerned implements and registers its operations at boot.

\* In `sysfs_fill_super()`, `sysfs_test_super()` and `sysfs_kill_sb()`, add loops to go through all registered operations structs and calls the corresponding operations if it's present.

But... I thought it was a bit overkill for the few namespaces that will actually need sysfs tagged directories.

(Below you'll find the traditional introduction for sysfs tagged dirs and the updated changelog)

Thanks,  
Benjamin

--

Here is an updated version of Eric Biederman's patchset to implement tagged directories in sysfs ported on top of 2.6.26-rc2-mm1.

With the introduction of network namespaces, there can be duplicate network interface names on the same machine. Indeed, two network interfaces can have the same name if they reside in different network namespaces.

- \* Network interfaces names show up in sysfs.
- \* Today there is nothing in sysfs that is currently per namespace.
- \* Therefore we need to support multiple mounts of sysfs each showing a different network namespace.

We introduce tagged directories in sysfs for this purpose.

Of course the usefulness of this feature is not limited to network stuff: Serge Hallyn wrote a patch to fix a similar issue with user namespaces based on this patchset. His patch is included at the end of the patchset.

Tested with and without `SYSFS_DEPRECATED`. No regression found so far.

## Changelog

-----

- \* V5:
  - Make namespace tags a bit less intrusive in sysfs core:
  - New patch 09: Added a generic `sysfs_ns_exit` routine called by exiting namespaces. A callback is passed to this routine to execute the subsystem specific code.
  - Modified patches 09 and 10 (now 10 and 11) ("netns tagging" and "usersns tagging") to use this new routine instead of adding `#ifdef'd` code in `fs/sysfs/mount.c`.
  - Added missing `-ENOMEM` in `fs/sysfs/dir.c:prep_rename()` (Roel Kluin)
- \* V4:

- Ported to 2.6.26-rc2-mm1
  - Updated patch for user namespace by Serge Hallyn (patch 10).
- \* V3:
- Removed patch 10 ("avoid kobject name conflict with different namespaces"), a better one was provided by Eric.
  - Removed patch 11 ("sysfs: user namespaces: add ns to user\_struct"), Serge needs to rework some parts of it.
  - Change Acked-by: to Signed-off-by:, someone told me it is more appropriate (as I'm in the delivery path).

Here is the announcement Eric wrote back in December to introduce his patchset:

"

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

[...]

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.

"

Regards,  
Benjamin

--

---

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

---

Subject: [PATCH 01/11] sysfs: Support for preventing unmounts.  
Posted by [Benjamin Thery](#) on Fri, 06 Jun 2008 15:47:04 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

sysfs: Support for preventing unmounts.

To support mounting multiple instances of sysfs occasionally 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>

Signed-off-by: Benjamin Thery <benjamin.thery@bull.net>

---

```
fs/sysfs/mount.c | 79 ++++++-----
fs/sysfs/sysfs.h | 10 ++++++
2 files changed, 81 insertions(+), 8 deletions(-)
```

Index: linux-mm/fs/sysfs/mount.c

=====

--- linux-mm.orig/fs/sysfs/mount.c

+++ linux-mm/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",__func__);
- 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;
Index: linux-mm/fs/sysfs/sysfs.h
=====
--- linux-mm.orig/fs/sysfs/sysfs.h
+++ linux-mm/fs/sysfs/sysfs.h
@@ -85,6 +85,12 @@ 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
+ */
@@ -92,6 +98,10 @@ extern struct sysfs_dirent sysfs_root;
extern struct super_block *sysfs_sb;
extern struct kmem_cache *sysfs_dir_cachep;
extern struct vfsmount *sysfs_mount;
+extern struct file_system_type sysfs_fs_type;
+
+void sysfs_grab_supers(void);
+void sysfs_release_supers(void);

/*
 * dir.c

--

```

---

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

---



---

Subject: [PATCH 02/11] sysfs: sysfs\_get\_dentry add a sb parameter  
Posted by [Benjamin Thery](#) on Fri, 06 Jun 2008 15:47:14 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

sysfs: sysfs\_get\_dentry add a sb parameter

In preparation for multiple mounts of sysfs add a superblock parameter to sysfs\_get\_dentry.

Signed-off-by: Eric W. Biederman <[ebiederm@xmission.com](mailto:ebiederm@xmission.com)>  
Signed-off-by: Benjamin Thery <[benjamin.thery@bull.net](mailto:benjamin.thery@bull.net)>

```

---
fs/sysfs/dir.c | 12 ++++++-----
fs/sysfs/file.c |  2 +-
fs/sysfs/sysfs.h |  3 ++-
3 files changed, 10 insertions(+), 7 deletions(-)

```

Index: linux-mm/fs/sysfs/dir.c

```

=====
--- linux-mm.orig/fs/sysfs/dir.c
+++ linux-mm/fs/sysfs/dir.c
@@ -87,6 +87,7 @@ static void sysfs_unlink_sibling(struct

/**
 * 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
@@ -99,9 +100,10 @@ static void sysfs_unlink_sibling(struct
 * 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;
@@ -790,7 +792,7 @@ int sysfs_rename_dir(struct kobject * ko
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);
old_dentry = NULL;
@@ -858,7 +860,7 @@ int sysfs_move_dir(struct kobject *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);
old_dentry = NULL;
@@ -866,7 +868,7 @@ int sysfs_move_dir(struct kobject *kobj,
}
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);
    new_parent = NULL;
}
```

Index: linux-mm/fs/sysfs/file.c

```
=====
--- linux-mm.orig/fs/sysfs/file.c
+++ linux-mm/fs/sysfs/file.c
@@ -584,7 +584,7 @@ int sysfs_chmod_file(struct kobject *kob
    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);
```

Index: linux-mm/fs/sysfs/sysfs.h

```
=====
--- linux-mm.orig/fs/sysfs/sysfs.h
+++ linux-mm/fs/sysfs/sysfs.h
@@ -113,7 +113,8 @@ 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,
```

--

---

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

---

Subject: [PATCH 03/11] sysfs: Implement \_\_sysfs\_get\_dentry  
Posted by [Benjamin Thery](#) on Fri, 06 Jun 2008 15:47:24 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

sysfs: Implement \_\_sysfs\_get\_dentry

This function is similar but much simpler to sysfs\_get\_dentry  
returns a sysfs dentry if one currently exists.

Signed-off-by: Eric W. Biederman <[ebiederm@xmission.com](mailto:ebiederm@xmission.com)>

Signed-off-by: Benjamin Thery <benjamin.thery@bull.net>

---

fs/sysfs/dir.c | 39 +++++  
1 file changed, 39 insertions(+)

Index: linux-mm/fs/sysfs/dir.c

=====

--- linux-mm.orig/fs/sysfs/dir.c

+++ linux-mm/fs/sysfs/dir.c

```
@@ -777,6 +777,45 @@ void sysfs_remove_dir(struct kobject * k
    __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;

--
```

---

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

---

---

Subject: [PATCH 04/11] sysfs: Rename Support multiple superblocks  
Posted by [Benjamin Thery](#) on Fri, 06 Jun 2008 15:47:34 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

sysfs: Rename Support multiple superblocks

This patch modifies the sysfs\_rename\_dir and sysfs\_move\_dir routines to support multiple sysfs dentry tries rooted in different sysfs superblocks.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>  
Signed-off-by: Benjamin Thery <benjamin.thery@bull.net>

---

fs/sysfs/dir.c | 195 ++++++-----  
1 file changed, 137 insertions(+), 58 deletions(-)

Index: linux-mm/fs/sysfs/dir.c

=====

--- linux-mm.orig/fs/sysfs/dir.c

+++ linux-mm/fs/sysfs/dir.c

```
@@ -816,43 +816,113 @@ static struct dentry *__sysfs_get_dentry
    return dentry;
}
```

```
+struct sysfs_rename_struct {
+ struct list_head list;
+ struct dentry *old_dentry;
+ struct dentry *new_dentry;
+ struct dentry *old_parent;
+ struct dentry *new_parent;
+};
+
+static void post_rename(struct list_head *head)
```

```

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

```

```

+ srs->new_parent = dentry;
+
+ error = -ENOMEM;
+ dentry = d_alloc_name(srs->new_parent, name);
+ if (!dentry)
+   goto err_out;
+ srs->new_dentry = dentry;
+ }
+ return 0;
+
+err_out:
+ post_rename(head);
+ return error;
+}
+
int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
{
    struct sysfs_dirent *sd = kobj->sd;
- struct dentry *parent = NULL;
- struct dentry *old_dentry = NULL, *new_dentry = NULL;
+ struct list_head todo;
+ struct sysfs_rename_struct *srs;
+ struct inode *parent_inode = NULL;
    const char *dup_name = NULL;
    int error;

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

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

- /* get the original dentry */
- old_dentry = sysfs_get_dentry(sysfs_sb, sd);
- if (IS_ERR(old_dentry)) {
-     error = PTR_ERR(old_dentry);
-     old_dentry = NULL;
-     goto out;
- }
+ sysfs_grab_supers();
+ error = prep_rename(&todo, sd, sd->s_parent, new_name);
+ if (error)
+     goto out_release;

- parent = old_dentry->d_parent;
+ error = -ENOMEM;
+ mutex_lock(&sysfs_mutex);

```

```

+ parent_inode = sysfs_get_inode(sd->s_parent);
+ mutex_unlock(&sysfs_mutex);
+ if (!parent_inode)
+ goto out_release;

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

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

- error = -ENOMEM;
- new_dentry = d_alloc_name(parent, new_name);
- if (!new_dentry)
- goto out_unlock;
-
  /* rename kobject and sysfs_dirent */
  error = -ENOMEM;
  new_name = dup_name = kstrdup(new_name, GFP_KERNEL);
@@ -867,17 +937,21 @@ int sysfs_rename_dir(struct kobject * ko
  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;
}
@@ -886,10 +960,12 @@ int sysfs_move_dir(struct kobject *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;
@@ -898,26 +974,29 @@ int sysfs_move_dir(struct kobject *kobj,
if (sd->s_parent == new_parent_sd)
goto out; /* nothing to move */

- /* get dentries */
- old_dentry = sysfs_get_dentry(sysfs_sb, sd);
- if (IS_ERR(old_dentry)) {
- error = PTR_ERR(old_dentry);
- old_dentry = NULL;
- goto out;
- }
- old_parent = old_dentry->d_parent;
-
- new_parent = sysfs_get_dentry(sysfs_sb, new_parent_sd);
- if (IS_ERR(new_parent)) {
- error = PTR_ERR(new_parent);
- new_parent = NULL;
- goto out;
- }
+ sysfs_grab_supers();
+ error = prep_rename(&todo, sd, new_parent_sd, sd->s_name);
+ if (error)
+ goto out_release;
+
+ 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);
@@ -926,14 +1005,11 @@ again:
  if (sysfs_find_dirent(new_parent_sd, sd->s_name))
    goto out_unlock;

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

  /* Remove from old parent's list and insert into new parent's list. */
  sysfs_unlink_sibling(sd);
@@ -942,14 +1018,17 @@ 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;
}
```

--

---

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

---

---

Subject: [PATCH 05/11] sysfs: sysfs\_chmod\_file handle multiple superblocks  
Posted by [Benjamin Thery](#) on Fri, 06 Jun 2008 15:47:44 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

sysfs: sysfs\_chmod\_file handle multiple superblocks

Teach sysfs\_chmod\_file how to handle multiple sysfs superblocks.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>  
Signed-off-by: Benjamin Thery <benjamin.thery@bull.net>

---

```
fs/sysfs/file.c | 54 ++++++-----
1 file changed, 30 insertions(+), 24 deletions(-)
```

Index: linux-mm/fs/sysfs/file.c

```
=====
--- linux-mm.orig/fs/sysfs/file.c
+++ linux-mm/fs/sysfs/file.c
@@ -573,7 +573,8 @@ EXPORT_SYMBOL_GPL(sysfs_add_file_to_grou
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;
@@ -584,31 +585,36 @@ int sysfs_chmod_file(struct kobjekt *kob
    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;
+ 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;
+ }
+
+ 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);

- 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);
+ 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;
}

--
```

---

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

---

---

Subject: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.  
Posted by [Benjamin Thery](#) on Fri, 06 Jun 2008 15:47:54 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

sysfs: Implement sysfs tagged directory support.

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 but 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>

Signed-off-by: Benjamin Thery <benjamin.thery@bull.net>

---

```

fs/sysfs/bin.c      | 2
fs/sysfs/dir.c     | 185 ++++++-----
fs/sysfs/file.c    | 8 +-
fs/sysfs/group.c   | 4 -
fs/sysfs/inode.c   | 7 +
fs/sysfs/mount.c   | 44 ++++++---
fs/sysfs/symlink.c | 2
fs/sysfs/sysfs.h   | 17 +++++
include/linux/sysfs.h | 17 +++++
9 files changed, 257 insertions(+), 29 deletions(-)

```

Index: linux-mm/fs/sysfs/bin.c

=====

--- linux-mm.orig/fs/sysfs/bin.c

+++ linux-mm/fs/sysfs/bin.c

```
@@ -252,7 +252,7 @@ int sysfs_create_bin_file(struct kobject
```

```
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);
```

```
Index: linux-mm/fs/sysfs/dir.c
```

```
-----
--- linux-mm.orig/fs/sysfs/dir.c
```

```
+++ linux-mm/fs/sysfs/dir.c
```

```
@@ -103,8 +103,17 @@ static void sysfs_unlink_sibling(struct
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;
```

```
@@ -423,7 +432,11 @@ void sysfs_addrm_start(struct sysfs_addr
*/
```

```
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);
```

```
@@ -439,6 +452,9 @@ int sysfs_add_one(struct sysfs_addrm_cxt
```

```
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);

@@ -585,13 +601,18 @@ void sysfs_addrm_finish(struct sysfs_add
 * 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;
+ }

@@ -615,7 +636,7 @@ struct sysfs_dirent *sysfs_get_dirent(st
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);

@@ -681,13 +702,16 @@ static struct dentry * sysfs_lookup(stru
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) {
@@ -895,19 +919,24 @@ int sysfs_rename_dir(struct kobject * ko
    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);
@@ -920,7 +949,7 @@ int sysfs_rename_dir(struct kobject * ko
    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 */
@@ -935,6 +964,8 @@ int sysfs_rename_dir(struct kobject * ko

    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) {
@@ -942,6 +973,20 @@ int sysfs_rename_dir(struct kobject * ko
    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);
@@ -964,11 +1009,13 @@ int sysfs_move_dir(struct kobject *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)
@@ -1002,7 +1049,7 @@ 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;
@@ -1041,10 +1088,11 @@ static inline unsigned char dt_type(stru

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;
@@ -1062,6 +1110,8 @@ static int sysfs_readdir(struct file * f
  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))
@@ -1071,6 +1121,10 @@ static int sysfs_readdir(struct file * f
  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;
@@ -1091,3 +1145,106 @@ const struct file_operations sysfs_dir_o
  .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 appropriate 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;
+}
Index: linux-mm/fs/sysfs/file.c

```

```

=====
--- linux-mm.orig/fs/sysfs/file.c
+++ linux-mm/fs/sysfs/file.c
@@ -460,9 +460,9 @@ void sysfs_notify(struct kobject *k, cha
    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;

@@ -631,7 +631,7 @@ 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);
}

@@ -651,7 +651,7 @@ void sysfs_remove_file_from_group(struct
    else
        dir_sd = sysfs_get(kobj->sd);
    if (dir_sd) {
- sysfs_hash_and_remove(dir_sd, attr->name);
+ sysfs_hash_and_remove(kobj, dir_sd, attr->name);
        sysfs_put(dir_sd);
    }
}
Index: linux-mm/fs/sysfs/group.c
=====
--- linux-mm.orig/fs/sysfs/group.c
+++ linux-mm/fs/sysfs/group.c
@@ -23,7 +23,7 @@ static void remove_files(struct sysfs_di
    int i;

    for (i = 0, attr = grp->attrs; *attr; i++, 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, struct kobject *kobj,
@@ -39,7 +39,7 @@ static int create_files(struct sysfs_dir
    * visibility. Do this by first removing then

```

```

* re-adding (if required) the file */
if (update)
- sysfs_hash_and_remove(dir_sd, (*attr)->name);
+ sysfs_hash_and_remove(kobj, dir_sd, (*attr)->name);
  if (grp->is_visible) {
    mode = grp->is_visible(kobj, *attr, i);
    if (!mode)

```

Index: linux-mm/fs/sysfs/inode.c

```
=====
--- linux-mm.orig/fs/sysfs/inode.c
```

```
+++ linux-mm/fs/sysfs/inode.c
```

```
@@ -217,17 +217,20 @@ struct inode * sysfs_get_inode(struct sy
  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);

```

Index: linux-mm/fs/sysfs/mount.c

```
=====
--- linux-mm.orig/fs/sysfs/mount.c
```

```
+++ linux-mm/fs/sysfs/mount.c
```

```
@@ -75,6 +75,7 @@ static int sysfs_fill_super(struct super
  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)
```

```
Index: linux-mm/fs/sysfs/symlink.c
```

```
-----  
--- linux-mm.orig/fs/sysfs/symlink.c
```

```
+++ linux-mm/fs/sysfs/symlink.c
```

```
@@ -94,7 +94,7 @@ void sysfs_remove_link(struct kobject *
```

```
    else
```

```
        parent_sd = kobj->sd;
```

```
- sysfs_hash_and_remove(parent_sd, name);
```

```
+ sysfs_hash_and_remove(kobj, parent_sd, name);
```

```
}
```

```
static int sysfs_get_target_path(struct sysfs_dirent *parent_sd,
```

```
Index: linux-mm/fs/sysfs/sysfs.h
```

```
-----  
--- linux-mm.orig/fs/sysfs/sysfs.h
```

```
+++ linux-mm/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)
@@ -113,6 +119,13 @@ 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);
@@ -124,6 +137,7 @@ void sysfs_remove_one(struct sysfs_addrm
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);
@@ -155,7 +169,8 @@ static inline void sysfs_put(struct sysf
    */
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);

/*
Index: linux-mm/include/linux/sysfs.h
=====
--- linux-mm.orig/include/linux/sysfs.h
+++ linux-mm/include/linux/sysfs.h
@@ -78,6 +78,14 @@ 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 *),
@@ -117,6 +125,9 @@ void sysfs_remove_file_from_group(struct
void sysfs_notify(struct kobject *kobj, char *dir, char *attr);
void sysfs_printk_last_file(void);

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

#else /* CONFIG_SYSFS */
@@ -213,6 +224,12 @@ static inline void sysfs_notify(struct k
{
}

+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;
}

--

```

---

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

---



---

Subject: [PATCH 07/11] sysfs: Implement sysfs\_delete\_link and sysfs\_rename\_link  
Posted by [Benjamin Thery](#) on Fri, 06 Jun 2008 15:48:04 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

sysfs: Implement sysfs\_delete\_link and sysfs\_rename\_link

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>

Signed-off-by: Benjamin Thery <benjamin.thery@bull.net>

```
---
fs/sysfs/symlink.c | 31 ++++++
include/linux/sysfs.h | 17 ++++++
2 files changed, 48 insertions(+)
```

Index: linux-mm/fs/sysfs/symlink.c

```
=====
--- linux-mm.orig/fs/sysfs/symlink.c
+++ linux-mm/fs/sysfs/symlink.c
@@ -80,6 +80,21 @@ int sysfs_create_link(struct kobject * k
 }

/**
+ * 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.
@@ -97,6 +112,22 @@ void sysfs_remove_link(struct kobject *
```

```
sysfs_hash_and_remove(kobj, parent_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)
```

```
{
```

```
Index: linux-mm/include/linux/sysfs.h
```

```
=====
```

```
--- linux-mm.orig/include/linux/sysfs.h
```

```
+++ linux-mm/include/linux/sysfs.h
```

```
@@ -111,6 +111,12 @@ int __must_check sysfs_create_link(struct  
    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);
```

```
int sysfs_update_group(struct kobject *kobj,
```

```
@@ -192,6 +198,17 @@ static inline void sysfs_remove_link(str
```

```
{
```

```
}
```

```
+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)
{
--

```

---

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

---



---

Subject: [PATCH 08/11] driver core: Implement tagged directory support for device classes.  
Posted by [Benjamin Thery](#) on Fri, 06 Jun 2008 15:48:14 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

driver core: Implement tagged directory support for device classes.

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>  
Signed-off-by: Benjamin Thery <benjamin.thery@bull.net>

```

---
drivers/base/class.c | 30 ++++++-----
drivers/base/core.c | 41 ++++++-----
include/linux/device.h | 2 ++
3 files changed, 50 insertions(+), 23 deletions(-)

```

Index: linux-mm/drivers/base/class.c

```

=====
--- linux-mm.orig/drivers/base/class.c
+++ linux-mm/drivers/base/class.c
@@ -134,6 +134,17 @@ static void remove_class_attrs(struct cl
 }
 }

```

```

+static int class_setup_tagging(struct class *cls)
+{
+ const struct sysfs_tagged_dir_operations *tag_ops;
+
+ tag_ops = cls->tag_ops;
+ if (!tag_ops)
+ return 0;
+
+ return sysfs_enable_tagging(&cls->subsys.kobj, tag_ops);
+}
+
int class_register(struct class *cls)
{
int error;
@@ -162,11 +173,22 @@ int class_register(struct class *cls)
cls->subsys.kobj.ktype = &class_ktype;

error = kset_register(&cls->subsys);
- if (!error) {
- error = add_class_attrs(class_get(cls));
- 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)
Index: linux-mm/drivers/base/core.c
=====
--- linux-mm.orig/drivers/base/core.c
+++ linux-mm/drivers/base/core.c
@@ -617,6 +617,10 @@ static struct kobject *get_device_parent
kobject_put(k);
return NULL;

```

```

}
+ /* If we created a new class-directory setup tagging */
+ if (dev->class->tag_ops)
+ sysfs_enable_tagging(k, dev->class->tag_ops);
+
+ /* do not emit an uevent for this simple "glue" directory */
+ return k;
+ }
@@ -751,12 +755,13 @@ static void device_remove_class_symlinks

if (dev->kobj.parent != &dev->class->subsys.kobj &&
    device_is_not_partition(dev))
- 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 && device_is_not_partition(dev))
    sysfs_remove_link(&dev->kobj, "device");

- sysfs_remove_link(&dev->class->subsys.kobj, dev->bus_id);
+ sysfs_delete_link(&dev->class->subsys.kobj, &dev->kobj, dev->bus_id);
#endif

    sysfs_remove_link(&dev->kobj, "subsystem");
@@ -1282,6 +1287,15 @@ int device_rename(struct device *dev, ch
    strcpy(old_device_name, dev->bus_id, BUS_ID_SIZE);
    strcpy(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) {
    strcpy(dev->bus_id, old_device_name, BUS_ID_SIZE);
@@ -1290,24 +1304,13 @@ int device_rename(struct device *dev, ch

#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",
-     __func__, error);
- }
+ if (new_class_name)
+ error = sysfs_rename_link(&dev->parent->kobj,
+     &dev->kobj,
+     old_class_name,
+     new_class_name);
+ }
#endif

```

Index: linux-mm/include/linux/device.h

```

=====
--- linux-mm.orig/include/linux/device.h
+++ linux-mm/include/linux/device.h
@@ -199,6 +199,8 @@ struct class {
    int (*suspend)(struct device *dev, pm_message_t state);
    int (*resume)(struct device *dev);

+ const struct sysfs_tagged_dir_operations *tag_ops;
+
    struct pm_ops *pm;
};

--

```

---

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

---

Subject: [PATCH 09/11] sysfs: add sysfs\_ns\_exit routine  
Posted by [Benjamin Thery](#) on Fri, 06 Jun 2008 15:48:24 GMT  
[View Forum Message](#) <> [Reply to Message](#)

Add sysfs routine sysfs\_ns\_exit() to allow a namespace to go away while sysfs is still mounted.

The exiting namespace calls this routine and pass it a callback to be called for every sysfs superblocks present. The callback contains the necessary code to clean the superblock tag data associated with this namespace.

Signed-off-by: Benjamin Thery <benjamin.thery@bull.net>

```
---
fs/sysfs/mount.c | 21 ++++++
include/linux/sysfs.h | 8 ++++++
2 files changed, 29 insertions(+)
```

Index: linux-mm/fs/sysfs/mount.c

```
=====
--- linux-mm.orig/fs/sysfs/mount.c
+++ linux-mm/fs/sysfs/mount.c
@@ -181,6 +181,27 @@ restart:
     spin_unlock(&sb_lock);
 }

+/* Clean sysfs tags related to a given namespace when it exits */
+void sysfs_ns_exit(void (*func)(struct sysfs_tag_info *, void *), void *data)
+{
+ /* Allow the namespace to go away while sysfs is still mounted. */
+ struct super_block *sb;
+ mutex_lock(&sysfs_rename_mutex);
+ sysfs_grab_supers();
+ mutex_lock(&sysfs_mutex);
+ list_for_each_entry(sb, &sysfs_fs_type.fs_supers, s_instances) {
+
+ struct sysfs_super_info *info = sysfs_info(sb);
+ /* Call the cleaning routine provided by the namespace.
+ * data is the current namespace id passed by the namespace.
+ */
+ func(&info->tag, data);
+ }
+ mutex_unlock(&sysfs_mutex);
+ sysfs_release_supers();
+ mutex_unlock(&sysfs_rename_mutex);
+}
+
int __init sysfs_init(void)
{
    int err = -ENOMEM;
Index: linux-mm/include/linux/sysfs.h
=====
```



```

--- linux-mm.orig/include/linux/sysfs.h
+++ linux-mm/include/linux/sysfs.h
@@ -134,6 +134,9 @@ void sysfs_printk_last_file(void);
int sysfs_enable_tagging(struct kobject *kobj,
    const struct sysfs_tagged_dir_operations *tag_ops);

+void sysfs_ns_exit(void (*func)(struct sysfs_tag_info *, void *),
+ void *data);
+
+extern int __must_check sysfs_init(void);

#else /* CONFIG_SYSFS */
@@ -247,6 +250,11 @@ static inline int sysfs_enable_tagging(s
    return 0;
}

+static inline void sysfs_ns_exit(void (*func)(struct sysfs_tag_info *, void *),
+ void *data)
+{
+}
+
static inline int __must_check sysfs_init(void)
{
    return 0;
}
--

```

---

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

---

Subject: [PATCH 10/11] netns: Enable tagging for net\_class directories in sysfs  
Posted by [Benjamin Thery](#) on Fri, 06 Jun 2008 15:48:34 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

net: Enable tagging for net\_class directories in sysfs

The problem. Network devices show up in sysfs and with the network namespace active multiple devices with the same name can show up in the same directory, ouch!

To avoid that problem and allow existing applications in network namespaces to see the same interface that is currently presented in sysfs, this patch enables the tagging directory support in sysfs.

By using the network namespace pointers as tags to separate out the the sysfs directory entries we ensure that we don't have conflicts

in the directories and applications only see a limited set of the network devices.

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

Signed-off-by: Benjamin Thery <benjamin.thery@bull.net>

```
---
fs/sysfs/mount.c      | 8 ++++++++
include/linux/sysfs.h | 2 ++
net/Kconfig           | 2 +-
net/core/net-sysfs.c | 45 ++++++++++++++++++++++++++++++++++++++
4 files changed, 56 insertions(+), 1 deletion(-)
```

Index: linux-mm/fs/sysfs/mount.c

```
=====
--- linux-mm.orig/fs/sysfs/mount.c
+++ linux-mm/fs/sysfs/mount.c
@@ -16,6 +16,8 @@
#include <linux/mount.h>
#include <linux/pagemap.h>
#include <linux/init.h>
+#include <linux/nsproxy.h>
+#include <net/net_namespace.h>

#include "sysfs.h"

@@ -78,6 +80,7 @@ static int sysfs_fill_super(struct super
    root->d_sb = sb;
    sb->s_root = root;
    sb->s_fs_info = info;
+ info->tag.net_ns = hold_net(current->nsproxy->net_ns);
    return 0;

out_err:
@@ -95,6 +98,9 @@ static int sysfs_test_super(struct super
    struct sysfs_super_info *info = sysfs_info(sb);
    int found = 1;

+ if (task->nsproxy->net_ns != info->tag.net_ns)
+ found = 0;
+
    return found;
}

@@ -131,6 +137,8 @@ static void sysfs_kill_sb(struct super_b
    struct sysfs_super_info *info = sysfs_info(sb);

    kill_anon_super(sb);
+ if (info->tag.net_ns)
```

```
+ release_net(info->tag.net_ns);
  kfree(info);
}
```

Index: linux-mm/include/linux/sysfs.h

```
-----
--- linux-mm.orig/include/linux/sysfs.h
+++ linux-mm/include/linux/sysfs.h
@@ -19,6 +19,7 @@
```

```
struct kobject;
struct module;
+struct net;
```

```
/* FIXME
 * The *owner field is no longer used, but leave around
@@ -79,6 +80,7 @@ struct sysfs_ops {
};
```

```
struct sysfs_tag_info {
+ struct net *net_ns;
};
```

```
struct sysfs_tagged_dir_operations {
```

Index: linux-mm/net/Kconfig

```
-----
--- linux-mm.orig/net/Kconfig
+++ linux-mm/net/Kconfig
@@ -30,7 +30,7 @@ menu "Networking options"
config NET_NS
  bool "Network namespace support"
  default n
- depends on EXPERIMENTAL && !SYSFS && NAMESPACES
+ depends on EXPERIMENTAL && NAMESPACES
  help
    Allow user space to create what appear to be multiple instances
    of the network stack.
```

Index: linux-mm/net/core/net-sysfs.c

```
-----
--- linux-mm.orig/net/core/net-sysfs.c
+++ linux-mm/net/core/net-sysfs.c
@@ -13,7 +13,9 @@
#include <linux/kernel.h>
#include <linux/netdevice.h>
#include <linux/if_arp.h>
+#include <linux/nsproxy.h>
#include <net/sock.h>
+#include <net/net_namespace.h>
```

```

#include <linux/rtnetlink.h>
#include <linux/wireless.h>
#include <net/iw_handler.h>
@@ -385,6 +387,28 @@ static struct attribute_group wireless_g
};
#endif

+/*
+ * sysfs: allow the net namespace to go away while sysfs is still mounted.
+ */
+static void net_sysfs_net_exit_cb(struct sysfs_tag_info *tag_info, void *data)
+{
+ struct net *net = (struct net *)data;
+
+ if (tag_info->net_ns != net)
+ return;
+ release_net(tag_info->net_ns);
+ tag_info->net_ns = NULL;
+}
+
+void net_sysfs_net_exit(struct net *net)
+{
+ sysfs_ns_exit(net_sysfs_net_exit_cb, net);
+}
+
+static struct pernet_operations net_sysfs_ops = {
+ .exit = net_sysfs_net_exit,
+};
+
#endif /* CONFIG_SYSFS */

#ifdef CONFIG_HOTPLUG
@@ -421,6 +445,23 @@ static void netdev_release(struct device
    kfree((char *)dev - dev->padded);
}

+static const void *net_sb_tag(struct sysfs_tag_info *info)
+{
+ return info->net_ns;
+}
+
+static const void *net_kobject_tag(struct kobject *kobj)
+{
+ struct net_device *dev;
+ dev = container_of(kobj, struct net_device, dev.kobj);
+ return dev_net(dev);
+}
+

```

```

+static const struct sysfs_tagged_dir_operations net_tagged_dir_operations = {
+ .sb_tag = net_sb_tag,
+ .kobject_tag = net_kobject_tag,
+};
+
static struct class net_class = {
    .name = "net",
    .dev_release = netdev_release,
@@ -430,6 +471,7 @@ static struct class net_class = {
#ifdef CONFIG_HOTPLUG
    .dev_uevent = netdev_uevent,
#endif
+ .tag_ops = &net_tagged_dir_operations,
};

/* Delete sysfs entries but hold kobject reference until after all
@@ -477,5 +519,8 @@ void netdev_initialize_kobject(struct ne

int netdev_kobject_init(void)
{
#ifdef CONFIG_SYSFS
+ register_pernet_subsys(&net_sysfs_ops);
#endif
    return class_register(&net_class);
}

--

```

---

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

---



---

Subject: [PATCH 11/11] sysfs: user namespaces: fix bug with  
clone(CLONE\_NEWUSER) with fairsched  
Posted by [Benjamin Thery](#) on Fri, 06 Jun 2008 15:48:44 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Mark the /sys/kernel/uids directory to be tagged so that processes in  
different user namespaces can remount /sys and see their own uid  
listings.

Without this patch, having CONFIG\_FAIR\_SCHED=y makes user namespaces  
unusable, because when you  
clone(CLONE\_NEWUSER)  
it will auto-create the root userid and try to create  
/sys/kernel/uids/0. Since that already exists from the parent user  
namespace, the create fails, and the clone misleadingly ends up

returning -ENOMEM.

This patch fixes the issue by allowing each user namespace to remount /sys, and having /sys filter the /sys/kernel/uid/ entries by user namespace.

Signed-off-by: Serge Hallyn <serue@us.ibm.com>

Signed-off-by: Benjamin Thery <benjamin.thery@bull.net>

```
---
fs/sysfs/mount.c      | 3 +++
include/linux/sched.h | 1 +
include/linux/sysfs.h | 2 ++
include/linux/user_namespace.h | 1 +
kernel/user.c         | 21 ++++++
kernel/user_namespace.c | 13 ++++++
6 files changed, 40 insertions(+), 1 deletion(-)
```

Index: linux-mm/fs/sysfs/mount.c

```
=====
--- linux-mm.orig/fs/sysfs/mount.c
+++ linux-mm/fs/sysfs/mount.c
@@ -81,6 +81,7 @@ static int sysfs_fill_super(struct super
    sb->s_root = root;
    sb->s_fs_info = info;
    info->tag.net_ns = hold_net(current->nsproxy->net_ns);
+ info->tag.user_ns = current->nsproxy->user_ns;
    return 0;
```

```
out_err:
@@ -100,6 +101,8 @@ static int sysfs_test_super(struct super

    if (task->nsproxy->net_ns != info->tag.net_ns)
        found = 0;
+ if (task->nsproxy->user_ns != info->tag.user_ns)
+ found = 0;

    return found;
}
```

Index: linux-mm/include/linux/sched.h

```
=====
--- linux-mm.orig/include/linux/sched.h
+++ linux-mm/include/linux/sched.h
@@ -600,6 +600,7 @@ struct user_struct {
    /* Hash table maintenance information */
    struct hlist_node uidhash_node;
    uid_t uid;
+ struct user_namespace *user_ns;
```

```
#ifdef CONFIG_USER_SCHED
struct task_group *tg;
Index: linux-mm/include/linux/sysfs.h
```

```
=====
--- linux-mm.orig/include/linux/sysfs.h
```

```
+++ linux-mm/include/linux/sysfs.h
```

```
@@ -20,6 +20,7 @@
```

```
struct kobject;
struct module;
struct net;
+struct user_namespace;
```

```
/* FIXME
```

```
* The *owner field is no longer used, but leave around
```

```
@@ -81,6 +82,7 @@ struct sysfs_ops {
```

```
struct sysfs_tag_info {
struct net *net_ns;
+ struct user_namespace *user_ns;
};
```

```
struct sysfs_tagged_dir_operations {
```

```
Index: linux-mm/include/linux/user_namespace.h
```

```
=====
--- linux-mm.orig/include/linux/user_namespace.h
```

```
+++ linux-mm/include/linux/user_namespace.h
```

```
@@ -12,6 +12,7 @@
```

```
struct user_namespace {
struct kref kref;
struct hlist_head uidhash_table[UIDHASH_SZ];
+ struct kset *kset;
struct user_struct *root_user;
};
```

```
Index: linux-mm/kernel/user.c
```

```
=====
--- linux-mm.orig/kernel/user.c
```

```
+++ linux-mm/kernel/user.c
```

```
@@ -53,6 +53,7 @@ struct user_struct root_user = {
```

```
.files = ATOMIC_INIT(0),
.sigpending = ATOMIC_INIT(0),
.locked_shm = 0,
```

```
+ .user_ns = &init_user_ns,
```

```
#ifdef CONFIG_USER_SCHED
```

```
.tg = &init_task_group,
```

```
#endif
```

```
@@ -236,6 +237,23 @@ static void uids_release(struct kobject
return;
```

```

}

+static const void *usersns_sb_tag(struct sysfs_tag_info *info)
+{
+ return info->user_ns;
+}
+
+static const void *usersns_kobject_tag(struct kobject *kobj)
+{
+ struct user_struct *up;
+ up = container_of(kobj, struct user_struct, kobj);
+ return up->user_ns;
+}
+
+static struct sysfs_tagged_dir_operations usersns_tagged_dir_operations = {
+ .sb_tag = usersns_sb_tag,
+ .kobject_tag = usersns_kobject_tag,
+};
+
+static struct kobj_type usersns_ktype = {
+ .sysfs_ops = &kobj_sysfs_ops,
+ .default_attrs = usersns_attributes,
@@ -272,6 +290,8 @@ int __init usersns_sysfs_init(void)
+ if (!usersns_kset)
+ return -ENOMEM;

+ sysfs_enable_tagging(&usersns_kset->kobj, &usersns_tagged_dir_operations);
+
+ return usersns_user_create(&root_user);
+}

@@ -404,6 +424,7 @@ struct user_struct *alloc_uid(struct user
+ goto out_unlock;

+ new->uid = uid;
+ new->user_ns = ns;
+ atomic_set(&new->__count, 1);

+ if (sched_create_user(new) < 0)
Index: linux-mm/kernel/user_namespace.c
=====
--- linux-mm.orig/kernel/user_namespace.c
+++ linux-mm/kernel/user_namespace.c
@@ -22,7 +22,7 @@ static struct user_namespace *clone_user
+ struct user_struct *new_user;
+ int n;

- ns = kmalloc(sizeof(struct user_namespace), GFP_KERNEL);

```



```

+ ns = kzalloc(sizeof(struct user_namespace), GFP_KERNEL);
  if (!ns)
    return ERR_PTR(-ENOMEM);

@@ -66,11 +66,22 @@ struct user_namespace * copy_user_ns(int
  return new_ns;
}

+/* clear sysfs tag when user namespace exits */
+static void sysfs_userns_exit(struct sysfs_tag_info *tag_info, void *data)
+{
+ struct user_namespace *ns = (struct user_namespace *)data;
+
+ if (tag_info->user_ns != ns)
+ return;
+ tag_info->user_ns = NULL;
+}
+
void free_user_ns(struct kref *kref)
{
  struct user_namespace *ns;

  ns = container_of(kref, struct user_namespace, kref);
+ sysfs_ns_exit(sysfs_userns_exit, ns);
  release_uids(ns);
  kfree(ns);
}

--

```

---

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

---



---

Subject: Re: [PATCH 11/11] sysfs: user namespaces: fix bug with  
clone(CLONE\_NEWUSER) with fairsched  
Posted by [serue](#) on Tue, 10 Jun 2008 22:41:12 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Quoting Benjamin Thery (benjamin.thery@bull.net):

- > Mark the /sys/kernel/uids directory to be tagged so that processes in
- > different user namespaces can remount /sys and see their own uid
- > listings.
- >
- > Without this patch, having CONFIG\_FAIR\_SCHED=y makes user namespaces
- > unusable, because when you
- > clone(CLONE\_NEWUSER)

```

> it will auto-create the root userid and try to create
> /sys/kernel/uids/0. Since that already exists from the parent user
> namespace, the create fails, and the clone misleadingly ends up
> returning -ENOMEM.
>
> This patch fixes the issue by allowing each user namespace to remount
> /sys, and having /sys filter the /sys/kernel/uid/ entries by user
> namespace.
>
> Signed-off-by: Serge Hallyn <serue@us.ibm.com>
> Signed-off-by: Benjamin Thery <benjamin.thery@bull.net>
> ---
> fs/sysfs/mount.c          |  3 +++
> include/linux/sched.h     |  1 +
> include/linux/sysfs.h     |  2 ++
> include/linux/user_namespace.h |  1 +
> kernel/user.c             | 21 +++++++++++++++++++++
> kernel/user_namespace.c   | 13 ++++++++
> 6 files changed, 40 insertions(+), 1 deletion(-)
>
> Index: linux-mm/fs/sysfs/mount.c
> =====
> --- linux-mm.orig/fs/sysfs/mount.c
> +++ linux-mm/fs/sysfs/mount.c
> @@ -81,6 +81,7 @@ static int sysfs_fill_super(struct super
> sb->s_root = root;
> sb->s_fs_info = info;
> info->tag.net_ns = hold_net(current->nsproxy->net_ns);
> + info->tag.user_ns = current->nsproxy->user_ns;
> return 0;
>
> out_err:
> @@ -100,6 +101,8 @@ static int sysfs_test_super(struct super
>
> if (task->nsproxy->net_ns != info->tag.net_ns)
> found = 0;
> + if (task->nsproxy->user_ns != info->tag.user_ns)
> + found = 0;
>
> return found;
> }
> Index: linux-mm/include/linux/sched.h
> =====
> --- linux-mm.orig/include/linux/sched.h
> +++ linux-mm/include/linux/sched.h
> @@ -600,6 +600,7 @@ struct user_struct {
> /* Hash table maintenance information */
> struct hlist_node uidhash_node;

```

```

> uid_t uid;
> + struct user_namespace *user_ns;
>
> #ifdef CONFIG_USER_SCHED
> struct task_group *tg;
> Index: linux-mm/include/linux/sysfs.h
> =====
> --- linux-mm.orig/include/linux/sysfs.h
> +++ linux-mm/include/linux/sysfs.h
> @@ -20,6 +20,7 @@
> struct kobject;
> struct module;
> struct net;
> +struct user_namespace;
>
> /* FIXME
> * The *owner field is no longer used, but leave around
> @@ -81,6 +82,7 @@ struct sysfs_ops {
>
> struct sysfs_tag_info {
> struct net *net_ns;
> + struct user_namespace *user_ns;
> };
>
> struct sysfs_tagged_dir_operations {
> Index: linux-mm/include/linux/user_namespace.h
> =====
> --- linux-mm.orig/include/linux/user_namespace.h
> +++ linux-mm/include/linux/user_namespace.h
> @@ -12,6 +12,7 @@
> struct user_namespace {
> struct kref kref;
> struct hlist_head uidhash_table[UIDHASH_SZ];
> + struct kset *kset;

```

Hmm, funky. I don't know where this came from - or actually, I think it survived from the second of 4 or 5 quick attempts to figure out where to do the tagging for usersns...

I don't think the kset should be here :)

Otherwise, looks good, thanks Benjamin.

-serge

```

> struct user_struct *root_user;
> };
>

```

```

> Index: linux-mm/kernel/user.c
> =====
> --- linux-mm.orig/kernel/user.c
> +++ linux-mm/kernel/user.c
> @@ -53,6 +53,7 @@ struct user_struct root_user = {
> .files = ATOMIC_INIT(0),
> .sigpending = ATOMIC_INIT(0),
> .locked_shm = 0,
> + .user_ns = &init_user_ns,
> #ifdef CONFIG_USER_SCHED
> .tg = &init_task_group,
> #endif
> @@ -236,6 +237,23 @@ static void uids_release(struct kobject
> return;
> }
>
> +static const void *usersns_sb_tag(struct sysfs_tag_info *info)
> +{
> + return info->user_ns;
> +}
> +
> +static const void *usersns_kobject_tag(struct kobject *kobj)
> +{
> + struct user_struct *up;
> + up = container_of(kobj, struct user_struct, kobj);
> + return up->user_ns;
> +}
> +
> +static struct sysfs_tagged_dir_operations usersns_tagged_dir_operations = {
> + .sb_tag = usersns_sb_tag,
> + .kobject_tag = usersns_kobject_tag,
> +};
> +
> static struct kobj_type uids_ktype = {
> .sysfs_ops = &kobj_sysfs_ops,
> .default_attrs = uids_attributes,
> @@ -272,6 +290,8 @@ int __init uids_sysfs_init(void)
> if (!uids_kset)
> return -ENOMEM;
>
> + sysfs_enable_tagging(&uids_kset->kobj, &usersns_tagged_dir_operations);
> +
> return uids_user_create(&root_user);
> }
>
> @@ -404,6 +424,7 @@ struct user_struct *alloc_uid(struct use
> goto out_unlock;
>

```

```

> new->uid = uid;
> + new->user_ns = ns;
> atomic_set(&new->__count, 1);
>
> if (sched_create_user(new) < 0)
> Index: linux-mm/kernel/user_namespace.c
> =====
> --- linux-mm.orig/kernel/user_namespace.c
> +++ linux-mm/kernel/user_namespace.c
> @@ -22,7 +22,7 @@ static struct user_namespace *clone_user
> struct user_struct *new_user;
> int n;
>
> - ns = kmalloc(sizeof(struct user_namespace), GFP_KERNEL);
> + ns = kzalloc(sizeof(struct user_namespace), GFP_KERNEL);
> if (!ns)
> return ERR_PTR(-ENOMEM);
>
> @@ -66,11 +66,22 @@ struct user_namespace *copy_user_ns(int
> return new_ns;
> }
>
> +/* clear sysfs tag when user namespace exits */
> +static void sysfs_userns_exit(struct sysfs_tag_info *tag_info, void *data)
> +{
> + struct user_namespace *ns = (struct user_namespace *)data;
> +
> + if (tag_info->user_ns != ns)
> + return;
> + tag_info->user_ns = NULL;
> +}
> +
> void free_user_ns(struct kref *kref)
> {
> struct user_namespace *ns;
>
> ns = container_of(kref, struct user_namespace, kref);
> + sysfs_ns_exit(sysfs_userns_exit, ns);
> release_uids(ns);
> kfree(ns);
> }
>
> --

```

---

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

---

Subject: Re: [PATCH 00/11] sysfs tagged directories V5

Posted by [serue](#) on Tue, 10 Jun 2008 22:50:24 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Quoting Benjamin Thery (benjamin.thery@bull.net):

> Greg,

>

> Here is an updated version of the sysfs tagged directories that improves  
> a bit the situation over the previous one.

>

> I've modified the patch 09 ("Enable tagging for net\_class directories in  
> sysfs") to be a bit less intrusive in sysfs core. I removed the #ifdef'd  
> parts you didn't like in fs/sysfs/mount.c, and replaced it by a generic  
> routine sysfs\_ns\_exit() that is called, if needed, by the namespace when  
> it exits. This routine goes through every sysfs super blocks and calls  
> the callback passed by the namespace to clean its tag.

>

> The patch is now splitted in two:

> \* 09/11: the generic routine,

> \* 10/11: the remaining network parts.

>

> The generic part can be merge with patch 05 ("sysfs: Implement sysfs  
> tagged directory support.") but I left it separate for now to ease  
> reviews.

Thanks. Not nearly as radical as I'd feared.

Greg, does this address your concern? It stops the need to put any non-braindead code (i.e. the \_exit notifier stuff) in fs/sysfs/mount.c for any new namespaces needing sysfs tagging, leaving only simple comparison/assignments.

> Serge's patch for user namespace is modified to use this new service too.

> No more #ifdef CONFIG\_NET or #ifdef CONFIG\_USER\_NS in fs/sysfs/mount.c  
> now.

>

>

> But, currently, a new namespace that wants to add its tag to sysfs dirs  
> still need to modify fs/sysfs/mount.c in a few routines to manage the  
> new tag member added in struct sysfs\_tag\_info: sysfs\_fill\_super(),  
> sysfs\_test\_super(), sysfs\_kill\_sb() (see the last two patches).  
> These changes are only the initialization and a bunch of comparisons.

>

> If we really want to go further, to get rid of these, I've thought  
> about:

>

> \* Extending sysfs\_tagged\_dir\_operations with super blocks operations:

> - fill\_sb\_tag, test\_sb\_tag, kill\_sb\_tag

>

> \* Add routines in sysfs to allow registration/unregistration of these  
> operations structs in a list:  
> - sysfs\_register\_tagged\_dir\_ops()...  
>  
> \* Each subsystem concerned implements and registers its operations at  
> boot.  
>  
> \* In sysfs\_fill\_super(), sysfs\_test\_super() and sysfs\_kill\_sb(), add  
> loops to go through all registered operations structs and calls the  
> corresponding operations if it's present.  
>  
> But... I thought it was a bit overkill for the few namespaces that will  
> actually need sysfs tagged directories.

Agreed. We expect device namespaces... anything else?

thanks,  
-serge

---

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

---

---

Subject: Re: [PATCH 00/11] sysfs tagged directories V5  
Posted by [gregkh](#) on Tue, 10 Jun 2008 23:00:58 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, Jun 10, 2008 at 05:50:24PM -0500, Serge E. Hallyn wrote:  
> Quoting Benjamin Thery (benjamin.thery@bull.net):  
> > Greg,  
> >  
> > Here is an updated version of the sysfs tagged directories that improves  
> > a bit the situation over the previous one.  
> >  
> > I've modified the patch 09 ("Enable tagging for net\_class directories in  
> > sysfs") to be a bit less intrusive in sysfs core. I removed the #ifdef'd  
> > parts you didn't like in fs/sysfs/mount.c, and replaced it by a generic  
> > routine sysfs\_ns\_exit() that is called, if needed, by the namespace when  
> > it exits. This routine goes through every sysfs super blocks and calls  
> > the callback passed by the namespace to clean its tag.  
> >  
> > The patch is now splitted in two:  
> > \* 09/11: the generic routine,  
> > \* 10/11: the remaining network parts.  
> >  
> > The generic part can be merge with patch 05 ("sysfs: Implement sysfs  
> > tagged directory support.") but I left it separate for now to ease

> > reviews.  
>  
> Thanks. Not nearly as radical as I'd feared.  
>  
> Greg, does this address your concern? It stops the need to put any  
> non-braindead code (i.e. the `_exit` notifier stuff) in `fs/sysfs/mount.c` for  
> any new namespaces needing `sysfs` tagging, leaving only simple  
> comparison/assignments.

Sorry, I've been swamped with other things this week and haven't gotten a chance to look this over. Hopefully by the end of the week... It's in my queue though, don't worry it will not get lost.

thanks,

greg k-h

---

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

---

---

Subject: Re: [PATCH 05/11] `sysfs`: `sysfs_chmod_file` handle multiple superblocks  
Posted by [Tejun Heo](#) on Wed, 25 Jun 2008 12:31:49 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Daniel Lezcano wrote:

> Tejun Heo wrote:

>> Hello,

>>

>> Daniel Lezcano wrote:

>>>> I think it would be great if `sysfs_chmod_file` can do all-or-nothing

>>>> instead of failing half way through but given the interface of

>>>> `notify_change()`, it could be difficult to implement. Any ideas?

>>> Is it acceptable to queue the notifications in a list until we are in

>>> the loop and loop again to notify when exiting the first loop without

>>> error ?

>>

>> Can you please take a look at the following patch?

>>

>> <http://article.gmane.org/gmane.linux.file-systems/24484>

>>

>> Which replaces `notify_change()` call to two calls to `sysfs_setattr()` and

>> `fsnotify_change()`. The latter never fails and the former should always

>> succeed if `inode_change_ok()` succeeds (`inode_setattr()` never fails

>> unless the size is changing), so I think the correct thing to do is...

>>

>> \* Separate out `sysfs_do_setattr()` which doesn't do `inode_change_ok()` and



>> just sets the attributes. Making it a void function which triggers  
>> WARN\_ON() when inode\_setattr() fails would be a good idea.  
>>  
>> \* Implement sysfs\_chmod\_file() in similar way rename/move are  
>> implemented - allocate all resources and check conditions and then iff  
>> everything looks okay commit the operation by calling sysfs\_do\_setattr().  
>>  
>> How does that sound?  
>  
> Does this patch looks like what you are describing ?

Yeah, something like that. With looping for all the inodes added, it looks like it will work fine.

Thanks.

--  
tejun

---

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

---