Subject: [PATCH 00/11] sysfs tagged directories V6
Posted by Benjamin Thery on Wed, 18 Jun 2008 17:07:29 GMT
View Forum Message <> Reply to Message

Greg, Andrew,

Here is yet another updated version of Eric Biederman's patchset to
implement tagged directories in sysfs ported on top of 2.6.26-rc5-mm3.
This may be easier for you to review as 2.6.26-rc2-mm1 was getting a
bit old now.

There is no major changes since the last version which introduced some
changes to address concerns from Greg about the patch "Enable tagging
for net_class directories" being to intrusive in sysfs core.

Refer to the Changelog below and
http://thread.gmane.org/gmane.linux.kernel/690799 to read the details
about the proposed changes (I didn't copy them here to save some space).

Andrew,

Can you consider merging this patchset in -mm until Greg has time
to re-review it and take it (or reject it)?

Thanks,
Benjamin

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

--

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
----
* V6:
  - Ported to 2.6.26-rc5-mm3
  - Patch 11 (userns) Removed an unused kset member from struct
    user_namespace left from a previous version of the patch.
* 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
      "userns 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

--

Subject: [PATCH 01/11] sysfs: Support for preventing unmounts.
Posted by Benjamin Thery on Wed, 18 Jun 2008 17:07:39 GMT

View Forum Message <> Reply to Message

sysfs: Support for preventing unmounts.

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>
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
```

--

Subject: [PATCH 02/11] sysfs: sysfs_get_dentry add a sb parameter
Posted by Benjamin Thery on Wed, 18 Jun 2008 17:07:49 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>
Signed-off-by: Benjamin Thery <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
@@ -85,6 +85,7 @@ 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
@@ -97,9 +98,10 @@ 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;
@@ -777,7 +779,7 @@ 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;
@@ -845,7 +847,7 @@ 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;
@@ -853,7 +855,7 @@ 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 Wed, 18 Jun 2008 17:07:59 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 curently exists.

Signed-off-by: Eric W. Biederman <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
@@ -764,6 +764,45 @@ 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 Wed, 18 Jun 2008 17:08:09 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
@@ -803,43 +803,113 @@ 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);
@@ -854,17 +924,21 @@ 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;
 }
@@ -873,10 +947,12 @@ 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;
@@ -885,26 +961,29 @@ 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);
@@ -913,14 +992,11 @@ 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);
@@ -929,14 +1005,17 @@ 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 Wed, 18 Jun 2008 17:08:19 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 kobject *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 Wed, 18 Jun 2008 17:08:29 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
@@ -101,8 +101,17 @@ 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;
@@ -421,11 +430,18 @@ 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))
   return -EEXIST;

  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);

@@ -572,13 +588,18 @@ 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;
 }

@@ -602,7 +623,7 @@ 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);

@@ -668,13 +689,16 @@ 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) {
@@ -882,19 +906,24 @@ 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);
@@ -907,7 +936,7 @@ 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 */
@@ -922,6 +951,8 @@ 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) {
@@ -929,6 +960,20 @@ 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);
@@ -951,11 +996,13 @@ 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)
@@ -989,7 +1036,7 @@ 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;
@@ -1028,10 +1075,11 @@ 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;
@@ -1049,6 +1097,8 @@ 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))
@@ -1058,6 +1108,10 @@ 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;
@@ -1078,3 +1132,106 @@ 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 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;
+}
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
@@ -80,6 +80,14 @@ 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 *),
@@ -119,6 +127,9 @@ 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 */
@@ -215,6 +226,12 @@ 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 Wed, 18 Jun 2008 17:08:39 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
@@ -113,6 +113,12 @@ int __must_check sysfs_create_link(struc
        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,
@@ -194,6 +200,17 @@ 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 Wed, 18 Jun 2008 17:08:49 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, 51 insertions(+), 22 deletions(-)

Index: linux-mm/drivers/base/class.c
===================================================================
--- linux-mm.orig/drivers/base/class.c
+++ linux-mm/drivers/base/class.c
@@ -135,6 +135,17 @@ 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->p->class_subsys.kobj, tag_ops);
+}
+
 int __class_register(struct class *cls, struct lock_class_key *key)
 {
  struct class_private *cp;
@@ -171,13 +182,24 @@ int __class_register(struct class *cls,
  cls->p = cp;

  error = kset_register(&cp->class_subsys);
- if (error) {
-  kfree(cp);
-  return error;
- }
+ if (error)
+  goto out_free_cp;
+
+ error = class_setup_tagging(cls);
+ if (error)
+  goto out_unregister;
+
  error = add_class_attrs(class_get(cls));
  class_put(cls);
+ if (error)
+  goto out_unregister;
+out:
  return error;
+out_unregister:
+ kset_unregister(&cp->class_subsys);
+out_free_cp:
+ kfree(cp);
+ goto out;
 }
 EXPORT_SYMBOL_GPL(__class_register);

Index: linux-mm/drivers/base/core.c
===================================================================
--- linux-mm.orig/drivers/base/core.c
+++ linux-mm/drivers/base/core.c
@@ -618,6 +618,10 @@ 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;
  }
@@ -754,13 +758,14 @@ static void device_remove_class_symlinks

  if (dev->kobj.parent != &dev->class->p->class_subsys.kobj &&
      device_is_not_partition(dev))
-  sysfs_remove_link(&dev->class->p->class_subsys.kobj,
+  sysfs_delete_link(&dev->class->p->class_subsys.kobj, &dev->kobj,
       dev_name(dev));
 #else
  if (dev->parent && device_is_not_partition(dev))
   sysfs_remove_link(&dev->kobj, "device");

- sysfs_remove_link(&dev->class->p->class_subsys.kobj, dev_name(dev));
+ sysfs_delete_link(&dev->class->p->class_subsys.kobj, &dev->kobj,
+     dev_name(dev));
 #endif

  sysfs_remove_link(&dev->kobj, "subsystem");
@@ -1349,6 +1354,16 @@ int device_rename(struct device *dev, ch
  strlcpy(old_device_name, dev_name(dev), BUS_ID_SIZE);
  strlcpy(dev->bus_id, new_name, BUS_ID_SIZE);

+#ifndef CONFIG_SYSFS_DEPRECATED
+ if (dev->class &&
+     (dev->kobj.parent != &dev->class->p->class_subsys.kobj)) {
```

```
+  error = sysfs_rename_link(&dev->class->p->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);
@@ -1357,23 +1372,13 @@ 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) {
-  error = sysfs_create_link(&dev->class->p->class_subsys.kobj,
-       &dev->kobj, dev_name(dev));
-  if (error)
-   goto out;
-  sysfs_remove_link(&dev->class->p->class_subsys.kobj,
-     old_device_name);
+  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
@@ -195,6 +195,8 @@ 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;
 	struct class_private *p;
 };
```

--
____
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 Wed, 18 Jun 2008 17:08:59 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
@@ -136,6 +136,9 @@ 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 */
@@ -249,6 +252,11 @@ 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 Wed, 18 Jun 2008 17:09:09 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;

 extern int kobject_set_name(struct kobject *kobj, const char *name, ...)
        __attribute__((format(printf, 2, 3)));
@@ -81,6 +82,7 @@ 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
@@ -475,5 +517,8 @@ 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 Wed, 18 Jun 2008 17:09:19 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 ++
 kernel/user.c           |   21 +++++++++++++++++++++
 kernel/user_namespace.c |   13 ++++++++++++-
 5 files changed, 39 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
@@ -604,6 +604,7 @@ 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;

 extern int kobject_set_name(struct kobject *kobj, const char *name, ...)
        __attribute__((format(printf, 2, 3)));
@@ -83,6 +84,7 @@ struct sysfs_ops {

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

 struct sysfs_tagged_dir_operations {
Index: linux-mm/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 *userns_sb_tag(struct sysfs_tag_info *info)
+{
+ return info->user_ns;
+}
+
+static const void *userns_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 userns_tagged_dir_operations = {
+ .sb_tag = userns_sb_tag,
+ .kobject_tag = userns_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, &userns_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 01/11] sysfs: Support for preventing unmounts.
Posted by Dave Hansen on Wed, 18 Jun 2008 17:44:23 GMT

View Forum Message <> Reply to Message

On Wed, 2008-06-18 at 19:07 +0200, Benjamin Thery wrote:
> To support mounting multiple instances of sysfs occassionally I
> need to walk through all of the currently present sysfs super blocks.

I know you may have addressed this before, but I forgot and it didn't make it into the changelogs.

Why are you doing this again? It seems like an awfully blunt instrument.

-- Dave

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

Subject: Re: [PATCH 01/11] sysfs: Support for preventing unmounts.
Posted by ebiederm on Wed, 18 Jun 2008 20:12:27 GMT

View Forum Message <> Reply to Message

Dave Hansen <dave@linux.vnet.ibm.com> writes:

> On Wed, 2008-06-18 at 19:07 +0200, Benjamin Thery wrote:
>> To support mounting multiple instances of sysfs occassionally I
>> need to walk through all of the currently present sysfs super blocks.
>
> I know you may have addressed this before, but I forgot and it didn't
> make it into the changelogs.
>
> Why are you doing this again? It seems like an awfully blunt
> instrument.

So the fundamentals.

- The data in sysfs fundamentally changes behind the back of the
  VFS and we need to keep the VFS in sync. Essentially this is the
  distributed filesystem problem.

- In particular for sysfs_rename and sysfs_move_dir we need to support finding
  the dcache entries and calling d_move. So that the dcache does not
  get into an inconsistent state. Timeouts and invalidates like NFS
  uses are to be avoided if at all possible.

- Coming through the vfs we are guaranteed that the filesystem will
  not be unmounted while we have a reference on a dentry, and with
  multiple mounts we do not get that guarantee. Therefore to get that
  guarantee for all of the superblocks we need the blunt instrument.

- Since mount/unmount are rare blocking them is no big deal.

I believe any distributed filesystem that is together enough to tell
us about renames (so we can update the dcache) instead of doing the
NFS timeout will need the ability to block mount/unmount while it is
executing d_move.

Currently sysfs does not need to block mounts only because we perform
an internal mount and then never unmount sysfs.

Eric

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

Subject: Re: [PATCH 09/11] sysfs: add sysfs_ns_exit routine
Posted by ebiederm on Wed, 18 Jun 2008 20:19:37 GMT

View Forum Message <> Reply to Message

Benjamin Thery <benjamin.thery@bull.net> writes:

> 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.

My apologies for not looking at this earlier. This is a nice cleanup.
Thank you.

Eric

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

Subject: Re: [PATCH 01/11] sysfs: Support for preventing unmounts.
Posted by Benjamin Thery on Thu, 19 Jun 2008 08:54:49 GMT

View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Dave Hansen <dave@linux.vnet.ibm.com> writes:
>
>> On Wed, 2008-06-18 at 19:07 +0200, Benjamin Thery wrote:
>>> To support mounting multiple instances of sysfs occassionally I
>>> need to walk through all of the currently present sysfs super blocks.
>> I know you may have addressed this before, but I forgot and it didn't
>> make it into the changelogs.
>>

>> Why are you doing this again? It seems like an awfully blunt
>> instrument.
>
> So the fundamentals.
> - The data in sysfs fundamentally changes behind the back of the
>   VFS and we need to keep the VFS in sync. Essentially this is the
>   distributed filesystem problem.
>
> - In particular for sysfs_rename and sysfs_move_dir we need to support finding
>   the dcache entries and calling d_move. So that the dcache does not
>   get into an inconsistent state. Timeouts and invalidates like NFS
>   uses are to be avoided if at all possible.
>
> - Coming through the vfs we are guaranteed that the filesystem will
>   not be unmounted while we have a reference on a dentry, and with
>   multiple mounts we do not get that guarantee. Therefore to get that
>   guarantee for all of the superblocks we need the blunt instrument.
>
> - Since mount/unmount are rare blocking them is no big deal.
>
> I believe any distributed filesystem that is together enough to tell
> us about renames (so we can update the dcache) instead of doing the
> NFS timeout will need the ability to block mount/unmount while it is
> executing d_move.
>
> Currently sysfs does not need to block mounts only because we perform
> an internal mount and then never unmount sysfs.

Thanks Eric for detailing this.
I think you explained it in much better way than I could do.
You're the author of the patch after all ;-)

Benjamin

>
> Eric
>
>

--
Benjamin Thery - BULL/DT/Open Software R&D

http://www.bull.com

Subject: Re: [PATCH 01/11] sysfs: Support for preventing unmounts.
Posted by Dave Hansen on Thu, 19 Jun 2008 16:32:11 GMT
View Forum Message <> Reply to Message

On Wed, 2008-06-18 at 13:12 -0700, Eric W. Biederman wrote:
>
> - The data in sysfs fundamentally changes behind the back of the
>   VFS and we need to keep the VFS in sync. Essentially this is the
>   distributed filesystem problem.
>
> - In particular for sysfs_rename and sysfs_move_dir we need to support
> finding
>   the dcache entries and calling d_move. So that the dcache does not
>   get into an inconsistent state. Timeouts and invalidates like NFS
>   uses are to be avoided if at all possible.

Much clearer now, thanks!

Can we get this description into the changelog, pretty please?

-- Dave

Subject: Re: [PATCH 01/11] sysfs: Support for preventing unmounts.
Posted by Benjamin Thery on Thu, 19 Jun 2008 20:19:41 GMT
View Forum Message <> Reply to Message

Quoting Dave Hansen <dave@linux.vnet.ibm.com>:

> On Wed, 2008-06-18 at 13:12 -0700, Eric W. Biederman wrote:
>>
>> - The data in sysfs fundamentally changes behind the back of the
>>   VFS and we need to keep the VFS in sync. Essentially this is the
>>   distributed filesystem problem.
>>
>> - In particular for sysfs_rename and sysfs_move_dir we need to support
>> finding
>>   the dcache entries and calling d_move. So that the dcache does not
>>   get into an inconsistent state. Timeouts and invalidates like NFS
>>   uses are to be avoided if at all possible.
>
> Much clearer now, thanks!
>

> Can we get this description into the changelog, pretty please?

Yes sure.
I'll add it to the patch introduction.

Benjamin

>
> -- Dave
>
>
>

This message was sent using IMP, the Internet Messaging Program.

____
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 Sun, 22 Jun 2008 04:46:50 GMT
View Forum Message <> Reply to Message

Hello, guys. Sorry about the long silence. Recent releases of
popular distros overwhelmed me with ata bugs. They now seem to be
under control (and hopefully stay that way for some time to come).

On the previous iteration, I was hoping I could sort out sysfs
interface problem before this patchset but given that this is a long
overdue feature, I think we should get this thing working first.

The first four patches looked good to me, so feel free to add
Acked-by: Tejun Heo <tj@kernel.org>

Benjamin Thery Wrote:
> sysfs: sysfs_chmod_file handle multiple superblocks
>
> Teach sysfs_chmod_file how to handle multiple sysfs
> superblocks.

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?

Thanks.

--
tejun

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by Tejun Heo on Mon, 23 Jun 2008 02:05:21 GMT

View Forum Message <> Reply to Message

Hello,

```
> 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;
>
```

As only directories can be tagged, I suppose handling tags explicitly
isn't necessary here, right? Can we please add a comment explaning
that?

```
> 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);
```

Taking both @kobj and @dir_sd is ugly but it isn't your fault. I'll clean things up later.

```
> Index: linux-mm/include/linux/sysfs.h
> ===================================================================
> --- linux-mm.orig/include/linux/sysfs.h
> +++ linux-mm/include/linux/sysfs.h
> @@ -80,6 +80,14 @@ 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);
> +};
```

As before, I can't bring myself to like this interface. Is computing tags dynamically really necessary? Can't we do the followings?

```
tag = sysfs_allocate_tag(s);
sysfs_enable_tag(kobj (or sd), tag);
sysfs_sb_show_tag(sb, tag);
```

Where tags are allocated using ida and each sb has bitmap of enabled tags so that sysfs ops can simply use something like the following to test whether it's enabled.

```
bool sysfs_tag_enabled(sb, tag)
```

```
{
 return sysfs_info(sb)->tag_map & (1 << tag);
}
```

Tags which can change dynamically seems too confusing to me and it
makes things difficult to verify as it's unclear how those tags are
gonna to change.

Thanks.

--
tejun

Subject: Re: [PATCH 07/11] sysfs: Implement sysfs_delete_link and sysfs_rename_link
Posted by Tejun Heo on Mon, 23 Jun 2008 02:13:12 GMT

View Forum Message <> Reply to Message

Benjamin Thery wrote:
> 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>

Ugly but given the current interface limitations...

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

--
tejun

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

Subject: Re: [PATCH 09/11] sysfs: add sysfs_ns_exit routine
Posted by Tejun Heo on Mon, 23 Jun 2008 02:16:12 GMT

View Forum Message <> Reply to Message

Benjamin Thery wrote:
> 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>

Similar object to earlier dynamic tag thing. Can't we just have
something like sysfs_murder_sb(sb) where @sb represents a ns?

--
tejun

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

Subject: Re: [PATCH 10/11] netns: Enable tagging for net_class directories in sysfs
Posted by Tejun Heo on Mon, 23 Jun 2008 02:18:12 GMT

View Forum Message <> Reply to Message

Benjamin Thery wrote:
```
> --- 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);
>  }
```

Ouch... Please don't hard code net_ns functions directly from sysfs.
Please make a proper abstraction for ns, make net_ns register it and
sysfs test the abstract ns.

Thanks.

--
tejun

Subject: Re: [PATCH 11/11] sysfs: user namespaces: fix bug with clone(CLONE_NEWUSER) with fairsched

Posted by Tejun Heo on Mon, 23 Jun 2008 02:18:58 GMT
View Forum Message <> Reply to Message

Benjamin Thery wrote:
> 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>

Ditto as patch #10.

Thanks.

--
tejun
____
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 Daniel Lezcano on Mon, 23 Jun 2008 21:42:35 GMT
View Forum Message <> Reply to Message

Tejun Heo wrote:
> Hello, guys. Sorry about the long silence. Recent releases of
> popular distros overwhelmed me with ata bugs. They now seem to be
> under control (and hopefully stay that way for some time to come).
>
> On the previous iteration, I was hoping I could sort out sysfs
> interface problem before this patchset but given that this is a long
> overdue feature, I think we should get this thing working first.
>

> The first four patches looked good to me, so feel free to add
> Acked-by: Tejun Heo <tj@kernel.org>
>
> Benjamin Thery Wrote:
>> sysfs: sysfs_chmod_file handle multiple superblocks
>>
>> Teach sysfs_chmod_file how to handle multiple sysfs
>> superblocks.
>
> 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 ?

____
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 Tue, 24 Jun 2008 04:45:27 GMT
View Forum Message <> Reply to Message

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?

Thanks.

--
tejun

Subject: Re: [PATCH 05/11] sysfs: sysfs_chmod_file handle multiple superblocks
Posted by Daniel Lezcano on Tue, 24 Jun 2008 10:39:46 GMT

View Forum Message <> Reply to Message

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?

Much better than my first proposition :)

I will do a separate patchset.

Thanks.

____
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 Wed, 25 Jun 2008 18:44:09 GMT

View Forum Message <> Reply to Message

Quoting Tejun Heo (htejun@gmail.com):
> Benjamin Thery wrote:
> > 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>
>
> Ditto as patch #10.

Except the sysfs mount holds no refcount on the userns. So as long as we
do the ida tagging as you suggested in your response to patch 6, there
should be no reference to the user_ns left in sysfs code.

The extra reference in patch #9 is for a light ref on the network

namespace. I'm still not sure that needs to be there, since if the
network namespace goes away, it will properly unregister its sysfs
mounts. Eric, Benjamin, I really don't see any use for the hold_net()
from sysfs. What is it doing?

-serge

____
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 ebiederm on Wed, 25 Jun 2008 21:11:39 GMT

View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Except the sysfs mount holds no refcount on the userns. So as long as we
> do the ida tagging as you suggested in your response to patch 6, there
> should be no reference to the user_ns left in sysfs code.
>
> The extra reference in patch #9 is for a light ref on the network
> namespace. I'm still not sure that needs to be there, since if the
> network namespace goes away, it will properly unregister its sysfs
> mounts. Eric, Benjamin, I really don't see any use for the hold_net()
> from sysfs. What is it doing?

Mostly just being a sanity check. We can remove that if it easier.

Eric

____
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 Thu, 26 Jun 2008 13:07:52 GMT

View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> > Except the sysfs mount holds no refcount on the userns. So as long as we

> > do the ida tagging as you suggested in your response to patch 6, there
> > should be no reference to the user_ns left in sysfs code.
> >
> > The extra reference in patch #9 is for a light ref on the network
> > namespace. I'm still not sure that needs to be there, since if the
> > network namespace goes away, it will properly unregister its sysfs
> > mounts. Eric, Benjamin, I really don't see any use for the hold_net()
> > from sysfs. What is it doing?
>
> Mostly just being a sanity check. We can remove that if it easier.
>
> Eric

In itself it seems an ok check (temporarily) to make sure that the
net_ns hook to unset the tag.netns at netns release is properly working,
but given that it's only checking for net_ns coding errors, and Tejun
wants a whole generic hooking infrastructure for net_ns to register with
so as to keep mention of net_ns out of sysfs code, yeah I think the sane
thing is to just remove it.

But the first thing (after Daniel is finished with the patch 5 fallout)
is to address the patch 6 comments about using ida. At first I didn't
like it (seemed like too much bookkeeping) but I think it'll actually
work out very nicely.

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

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by ebiederm on Thu, 26 Jun 2008 20:21:24 GMT

View Forum Message <> Reply to Message

Tejun thank you for the review, and my apologies for the delayed
reply.

Tejun Heo <htejun@gmail.com> writes:

```
>> Index: linux-mm/include/linux/sysfs.h
>> ===================================================================
>> --- linux-mm.orig/include/linux/sysfs.h
>> +++ linux-mm/include/linux/sysfs.h
>> @@ -80,6 +80,14 @@ 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);
>> +};
>
> As before, I can't bring myself to like this interface. Is computing
> tags dynamically really necessary? Can't we do the followings?

It isn't so much computing tags dynamically but rather it is reading them
from where they are stored.

> tag = sysfs_allocate_tag(s);
> sysfs_enable_tag(kobj (or sd), tag);
> sysfs_sb_show_tag(sb, tag);
>
> Where tags are allocated using ida and each sb has bitmap of enabled
> tags so that sysfs ops can simply use something like the following to
> test whether it's enabled.
>
> bool sysfs_tag_enabled(sb, tag)
> {
> return sysfs_info(sb)->tag_map & (1 << tag);
> }

Youch that seems limiting. The expectation is that we could have
as many as 100 different containers in use on a single system at one
time. So 100 apparent copies of the network stack.

There is also a second dimension here we multiplex different
directories based on different sets of tags.   One directory based
on user namespaces another on the network namespaces.

The tags in practice are just pointers to the namespace pointers.

So while we could use the ida technique to specify which set of tags
we are talking about for a directory it isn't sufficient.

The question sysfs_tag_enabled(sb, tag) makes no sense to me.
Especially in the context of needed a sysfs_sb_show_tag(sb, tag);

The current structure is because of all of the darn fool races and
magic that sysfs does. We have to say for a given directory: Your
contents will always be tagged, and only those that one tag that

matches what was captured by the superblock when sysfs is mounted
will be shown.

> Tags which can change dynamically seems too confusing to me and it
> makes things difficult to verify as it's unclear how those tags are
> gonna to change.

We have a fundamental issue that we have to handle, and it sounds like
you are proposing something that will not handle it.

- network devices can move between namespaces.
- network devices have driver specific sysfs attributes hanging off of them.

So we have to move the network devices and their sysfs attributes
between namespaces, and I implemented that in kobject_rename,
sysfs_rename path.

The tags on a kobject can only change during a rename operation.
So when the change happens is well defined. Further there is a
set of functions: sysfs_creation_tag, sysfs_removal_tag,
sysfs_lookup_tag, sysfs_dirent_tag which makes it clear what we
are doing.

If you really don't like how the tags are managed we need to talk
about how we store the tags on kobjects and on the super block.

Registering a set of tags could easily make the sb_tag function
obsolete, and that is one small piece of code so it is no big deal.

```
struct sysfs_tag_type_operations {
 const void *(*mount_tag)(void);
        const void *(*kobject_tag)(struct kobject *kobj);
};
```

Then we could do:

```
struct sysfs_sbtag_operations *tag_type_ops[MAX_TAG_TYPES];
```

And sysfs_tag_info could become.

```
struct sysfs_tag_info {
       void *tag[MAX_TAG_TYPES];
};
```

During subsystem initialization we could call

```
tag_type = sysfs_allocate_tag_type();
```

Just after the subsystem creates a directory.

```
sysfs_enable_tagging(kobj/sd, tag_type);
```

Then anytime we currently call sb_tag during lookup we can instead
just look at sysfs_info(sb)->tag[tag_type] and compare that with
sd->s_tag.tag.

The actual tag values themselves are current stored in the object in
which the kobject is embedded.

So we still need to call kobject_tag when we create or rename
something in a tagged directory. So we know what the tag is.

When we go to remove a kobj using the existing tag on the object
is the right choice.

Rename is the fun case where we need to grab the old tag from the
sd and place on it the new tag from kobject_tag.

One of the big problems at least with the class directories is that
the lifetimes are completely decoupled the between the tags and
the subsystem objects and subsystem directories that need to be
tagged. This isn't a set things up at the start of your subsystem
and everything is happy situation. To handle the races there must
be support at least at the kobject level for handling this in the
network namespace case.

Eric

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

Subject: Re: [PATCH 07/11] sysfs: Implement sysfs_delete_link and sysfs_rename_link
Posted by ebiederm on Thu, 26 Jun 2008 20:24:47 GMT

View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> Ugly but given the current interface limitations...

Yep. I have been thinking it might be nice to look at what you did with magic symlink
handling. Otherwise we either need to convert everything from sysfs_remove_link to
sysfs_delete_link if we don't want to play whack a mole.

Eric

____
Containers mailing list
Containers@lists.linux-foundation.org

https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 07/11] sysfs: Implement sysfs_delete_link and sysfs_rename_link
Posted by Tejun Heo on Sun, 29 Jun 2008 03:35:36 GMT

View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Tejun Heo <htejun@gmail.com> writes:
>
>> Ugly but given the current interface limitations...
>
> Yep. I have been thinking it might be nice to look at what you did with magic symlink
> handling. Otherwise we either need to convert everything from sysfs_remove_link to
> sysfs_delete_link if we don't want to play whack a mole.

I think the current one should do for now. Changing to managed links
requires pervasive changes to locking and stuff. I'll clean them up
when I refresh the patches I posted a while back.

Thanks.

--
tejun

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

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by Tejun Heo on Sun, 29 Jun 2008 03:51:37 GMT

View Forum Message <> Reply to Message

Hello, Eric.

Eric W. Biederman wrote:
> Tejun thank you for the review, and my apologies for the delayed
> reply.

Me being the king of delays, no need for apologies. :-)

>> As before, I can't bring myself to like this interface. Is computing
>> tags dynamically really necessary? Can't we do the followings?
>
> It isn't so much computing tags dynamically but rather it is reading them

> from where they are stored.

It's still dynamic from sysfs's POV and I think that will make
maintenance more difficult.

```
>> tag = sysfs_allocate_tag(s);
>> sysfs_enable_tag(kobj (or sd), tag);
>> sysfs_sb_show_tag(sb, tag);
>>
>> Where tags are allocated using ida and each sb has bitmap of enabled
>> tags so that sysfs ops can simply use something like the following to
>> test whether it's enabled.
>>
>> bool sysfs_tag_enabled(sb, tag)
>> {
>>  return sysfs_info(sb)->tag_map & (1 << tag);
>> }
>
>
```

> Youch that seems limiting. The expectation is that we could have
> as many as 100 different containers in use on a single system at one
> time. So 100 apparent copies of the network stack.

100 netns would mean 100 bits and 100 different views of them would mean
100 sb's where each sb would need bitmap larger than 100 bits. I don't
think there would be a scalability problem. Am I missing something?

> There is also a second dimension here we multiplex different
> directories based on different sets of tags.   One directory based
> on user namespaces another on the network namespaces.

No matter which criteria is used to select ns, it should end up being
mapped to a set of tags (here, ida allocated numbers). Unless tags can
change dynamically, there shouldn't be functional difference.

> The tags in practice are just pointers to the namespace pointers.
>
> So while we could use the ida technique to specify which set of tags
> we are talking about for a directory it isn't sufficient.

I failed to follow here. Can you please elaborate a bit? If you can
describe a simple example to me, it would be much appreciated.

> The question sysfs_tag_enabled(sb, tag) makes no sense to me.
> Especially in the context of needed a sysfs_sb_show_tag(sb, tag);
>
> The current structure is because of all of the darn fool races and
> magic that sysfs does. We have to say for a given directory: Your

> contents will always be tagged, and only those that one tag that
> matches what was captured by the superblock when sysfs is mounted
> will be shown.

sysfs_tag_enabled() was meant to test whether a directory which is
tagged should be shown under the current sb.

>> Tags which can change dynamically seems too confusing to me and it
>> makes things difficult to verify as it's unclear how those tags are
>> gonna to change.
>
> We have a fundamental issue that we have to handle, and it sounds like
> you are proposing something that will not handle it.
>
> - network devices can move between namespaces.
> - network devices have driver specific sysfs attributes hanging off of them.
>
> So we have to move the network devices and their sysfs attributes
> between namespaces, and I implemented that in kobject_rename,
> sysfs_rename path.
>
> The tags on a kobject can only change during a rename operation.
> So when the change happens is well defined. Further there is a
> set of functions: sysfs_creation_tag, sysfs_removal_tag,
> sysfs_lookup_tag, sysfs_dirent_tag which makes it clear what we
> are doing.
>
> If you really don't like how the tags are managed we need to talk
> about how we store the tags on kobjects and on the super block.
>
> Registering a set of tags could easily make the sb_tag function
> obsolete, and that is one small piece of code so it is no big deal.
>
> struct sysfs_tag_type_operations {
> const void *(*mount_tag)(void);
>   const void *(*kobject_tag)(struct kobject *kobj);
> };
>
> Then we could do:
> struct sysfs_sbtag_operations *tag_type_ops[MAX_TAG_TYPES];
>
> And sysfs_tag_info could become.
> struct sysfs_tag_info {
>   void *tag[MAX_TAG_TYPES];
> };
>
> During subsystem initialization we could call
> tag_type = sysfs_allocate_tag_type();

>
> Just after the subsystem creates a directory.
> sysfs_enable_tagging(kobj/sd, tag_type);
>
> Then anytime we currently call sb_tag during lookup we can instead
> just look at sysfs_info(sb)->tag[tag_type] and compare that with
> sd->s_tag.tag.

What you described is pretty much what I'm talking about. The only difference is whether to use caller-provided pointer as tag or an ida-allocated integer. The last sentence of the above paragraph is basically sys_tag_enabled() function (maybe misnamed).

The main reason why I'm whining about this so much is because I think tag should be something abstracted inside sysfs proper. It's something which affects very internal operation of sysfs and I really want to keep the implementation details inside sysfs. Spreading implementation over kobject and sysfs didn't turn out too pretty after all.

Thank you.

--
tejun

Subject: Re: [PATCH 07/11] sysfs: Implement sysfs_delete_link and sysfs_rename_link
Posted by ebiederm on Mon, 30 Jun 2008 03:02:22 GMT

View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:
>
> I think the current one should do for now. Changing to managed links
> requires pervasive changes to locking and stuff. I'll clean them up
> when I refresh the patches I posted a while back.

Sounds good.

Eric

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by ebiederm on Mon, 30 Jun 2008 18:56:44 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> Hello, Eric.
>
> Eric W. Biederman wrote:
>> Tejun thank you for the review, and my apologies for the delayed
>> reply.
>
> Me being the king of delays, no need for apologies. :-)
>
>>> As before, I can't bring myself to like this interface. Is computing
>>> tags dynamically really necessary? Can't we do the followings?
>>
>> It isn't so much computing tags dynamically but rather it is reading them
>> from where they are stored.
>
> It's still dynamic from sysfs's POV and I think that will make
> maintenance more difficult.

Potentially. I have no problem make it clear that things are more static.

>> There is also a second dimension here we multiplex different
>> directories based on different sets of tags.   One directory based
>> on user namespaces another on the network namespaces.
>
> No matter which criteria is used to select ns, it should end up being
> mapped to a set of tags (here, ida allocated numbers). Unless tags can
> change dynamically, there shouldn't be functional difference.
>
>> The tags in practice are just pointers to the namespace pointers.
>>
>> So while we could use the ida technique to specify which set of tags
>> we are talking about for a directory it isn't sufficient.
>
> I failed to follow here. Can you please elaborate a bit? If you can
> describe a simple example to me, it would be much appreciated.

See below.

>> The question sysfs_tag_enabled(sb, tag) makes no sense to me.
>> Especially in the context of needed a sysfs_sb_show_tag(sb, tag);
>>
>> The current structure is because of all of the darn fool races and
>> magic that sysfs does. We have to say for a given directory: Your
>> contents will always be tagged, and only those that one tag that

>> matches what was captured by the superblock when sysfs is mounted
>> will be shown.
>
> sysfs_tag_enabled() was meant to test whether a directory which is
> tagged should be shown under the current sb.

Ah. When we are doing readdir or lookup. Yes that makes sense.

See below. I honestly think sysfs_tab_enabled is the wrong question.

>>> Tags which can change dynamically seems too confusing to me and it
>>> makes things difficult to verify as it's unclear how those tags are
>>> gonna to change.
>>
>> We have a fundamental issue that we have to handle, and it sounds like
>> you are proposing something that will not handle it.
>>
>> - network devices can move between namespaces.
>> - network devices have driver specific sysfs attributes hanging off of them.
>>
>> So we have to move the network devices and their sysfs attributes
>> between namespaces, and I implemented that in kobject_rename,
>> sysfs_rename path.
>>
>> The tags on a kobject can only change during a rename operation.
>> So when the change happens is well defined. Further there is a
>> set of functions: sysfs_creation_tag, sysfs_removal_tag,
>> sysfs_lookup_tag, sysfs_dirent_tag which makes it clear what we
>> are doing.
>>
>> If you really don't like how the tags are managed we need to talk
>> about how we store the tags on kobjects and on the super block.
>>
>> Registering a set of tags could easily make the sb_tag function
>> obsolete, and that is one small piece of code so it is no big deal.
>>
>> struct sysfs_tag_type_operations {
>> const void *(*mount_tag)(void);
>>   const void *(*kobject_tag)(struct kobject *kobj);
>> };
>>
>> Then we could do:
>> struct sysfs_sbtag_operations *tag_type_ops[MAX_TAG_TYPES];
>>
>> And sysfs_tag_info could become.
>> struct sysfs_tag_info {
>>   void *tag[MAX_TAG_TYPES];
>> };

>>
>> During subsystem initialization we could call
>> tag_type = sysfs_allocate_tag_type();
>>
>> Just after the subsystem creates a directory.
>> sysfs_enable_tagging(kobj/sd, tag_type);
>>
>> Then anytime we currently call sb_tag during lookup we can instead
>> just look at sysfs_info(sb)->tag[tag_type] and compare that with
>> sd->s_tag.tag.
>
> What you described is pretty much what I'm talking about. The only
> difference is whether to use caller-provided pointer as tag or an
> ida-allocated integer. The last sentence of the above paragraph is
> basically sys_tag_enabled() function (maybe misnamed).

So some concrete code examples here. In the current code in lookup
what I am doing is:

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

With the proposed change of adding tag types sysfs_lookup_tag becomes:

```
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 = sysfs_info(sb)->tag[dir_sd->tag_type];

 return tag;
}
```

Which means that in practice I can lookup that tag that I am displaying
once.

Then in sysfs_find_dirent we do:

```
 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;
 }
```

That should keep the implementation sufficiently inside of sysfs for there

to be no guessing. In addition as a practical matter we can only allow
one tag to be visible in a directory at once or else we can not check
for duplicate names. Which is the problem I see with a bitmap based test
too unnecessary many degrees of freedom.

The number of tag types will be low as it is the number of subsystems
that use the feature. Simple enough that I expect statically allocating
the tag types in an enumeration is a safe and sane way to operate.
i.e.

```
enum sysfs_tag_types {
 SYSFS_TAG_NETNS,
 SYSFS_TAG_USERNS,
        SYSFS_TAG_MAX
};
```

> The main reason why I'm whining about this so much is because I think
> tag should be something abstracted inside sysfs proper. It's something
> which affects very internal operation of sysfs and I really want to keep
> the implementation details inside sysfs. Spreading implementation over
> kobject and sysfs didn't turn out too pretty after all.

I agree. Most of the implementation is in sysfs already. We just have
a few corner cases.

Fundamentally it is the subsystems responsibility that creates the
kobjects and the sysfs entries. The only case where I can see an
ida generated number being a help is if we start having lifetime
issues. Further the extra work to allocate and free tags ida based
tags seems unnecessary.

I don't doubt that there is a lot we can do better. My current goal
is for something that is clean enough it won't get us into trouble
later, and then merging the code. In tree where people can see
the code and the interactions I expect it will be easier to talk
about.

Currently the interface with the users is very small. Adding the
tag_type enumeration should make it smaller and make things more
obviously static.

Guys can we please make something useful happen?

Eric

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by serue on Mon, 30 Jun 2008 21:44:23 GMT

View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> Tejun Heo <htejun@gmail.com> writes:
>
> > Hello, Eric.
> >
> > Eric W. Biederman wrote:
> >> Tejun thank you for the review, and my apologies for the delayed
> >> reply.
> >
> > Me being the king of delays, no need for apologies. :-)
> >
> >>> As before, I can't bring myself to like this interface. Is computing
> >>> tags dynamically really necessary? Can't we do the followings?
> >>
> >> It isn't so much computing tags dynamically but rather it is reading them
> >> from where they are stored.
> >
> > It's still dynamic from sysfs's POV and I think that will make
> > maintenance more difficult.
>
> Potentially. I have no problem make it clear that things are more static.
>
> >> There is also a second dimension here we multiplex different
> >> directories based on different sets of tags.   One directory based
> >> on user namespaces another on the network namespaces.
> >
> > No matter which criteria is used to select ns, it should end up being
> > mapped to a set of tags (here, ida allocated numbers). Unless tags can
> > change dynamically, there shouldn't be functional difference.
> >
> >> The tags in practice are just pointers to the namespace pointers.
> >>
> >> So while we could use the ida technique to specify which set of tags
> >> we are talking about for a directory it isn't sufficient.
> >
> > I failed to follow here. Can you please elaborate a bit? If you can
> > describe a simple example to me, it would be much appreciated.
>
> See below.
>
> >> The question sysfs_tag_enabled(sb, tag) makes no sense to me.
> >> Especially in the context of needed a sysfs_sb_show_tag(sb, tag);
> >>
> >> The current structure is because of all of the darn fool races and
> >> magic that sysfs does. We have to say for a given directory: Your

> >> contents will always be tagged, and only those that one tag that
> >> matches what was captured by the superblock when sysfs is mounted
> >> will be shown.
> >
> > sysfs_tag_enabled() was meant to test whether a directory which is
> > tagged should be shown under the current sb.
>
> Ah. When we are doing readdir or lookup. Yes that makes sense.
>
> See below. I honestly think sysfs_tab_enabled is the wrong question.
>
> >>> Tags which can change dynamically seems too confusing to me and it
> >>> makes things difficult to verify as it's unclear how those tags are
> >>> gonna to change.
> >>
> >> We have a fundamental issue that we have to handle, and it sounds like
> >> you are proposing something that will not handle it.
> >>
> >> - network devices can move between namespaces.
> >> - network devices have driver specific sysfs attributes hanging off of them.
> >>
> >> So we have to move the network devices and their sysfs attributes
> >> between namespaces, and I implemented that in kobject_rename,
> >> sysfs_rename path.
> >>
> >> The tags on a kobject can only change during a rename operation.
> >> So when the change happens is well defined. Further there is a
> >> set of functions: sysfs_creation_tag, sysfs_removal_tag,
> >> sysfs_lookup_tag, sysfs_dirent_tag which makes it clear what we
> >> are doing.
> >>
> >> If you really don't like how the tags are managed we need to talk
> >> about how we store the tags on kobjects and on the super block.
> >>
> >> Registering a set of tags could easily make the sb_tag function
> >> obsolete, and that is one small piece of code so it is no big deal.
> >>
> >> struct sysfs_tag_type_operations {
> >> const void *(*mount_tag)(void);
> >>   const void *(*kobject_tag)(struct kobject *kobj);
> >> };
> >>
> >> Then we could do:
> >> struct sysfs_sbtag_operations *tag_type_ops[MAX_TAG_TYPES];
> >>
> >> And sysfs_tag_info could become.
> >> struct sysfs_tag_info {
> >>   void *tag[MAX_TAG_TYPES];

```
> >> };
> >>
> >> During subsystem initialization we could call
> >> tag_type = sysfs_allocate_tag_type();
> >>
> >> Just after the subsystem creates a directory.
> >> sysfs_enable_tagging(kobj/sd, tag_type);
> >>
> >> Then anytime we currently call sb_tag during lookup we can instead
> >> just look at sysfs_info(sb)->tag[tag_type] and compare that with
> >> sd->s_tag.tag.
> >
> > What you described is pretty much what I'm talking about.  The only
> > difference is whether to use caller-provided pointer as tag or an
> > ida-allocated integer.  The last sentence of the above paragraph is
> > basically sys_tag_enabled() function (maybe misnamed).
>
> So some concrete code examples here.  In the current code in lookup
> what I am doing is:
>
>  tag = sysfs_lookup_tag(parent_sd, parent->d_sb);
>  sd = sysfs_find_dirent(parent_sd, tag, dentry->d_name.name);
>
> With the proposed change of adding tag types sysfs_lookup_tag becomes:
>
> 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 = sysfs_info(sb)->tag[dir_sd->tag_type];
>
>  return tag;
> }
>
> Which means that in practice I can lookup that tag that I am displaying
> once.
>
> Then in sysfs_find_dirent we do:
>
>  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;
>  }
>
```

> That should keep the implementation sufficiently inside of sysfs for there
> to be no guessing. In addition as a practical matter we can only allow
> one tag to be visible in a directory at once or else we can not check
> for duplicate names. Which is the problem I see with a bitmap based test
> too unnecessary many degrees of freedom.
>
> The number of tag types will be low as it is the number of subsystems
> that use the feature. Simple enough that I expect statically allocating
> the tag types in an enumeration is a safe and sane way to operate.
> i.e.
>
> enum sysfs_tag_types {
> SYSFS_TAG_NETNS,
> SYSFS_TAG_USERNS,
>   SYSFS_TAG_MAX
> };
>
> > The main reason why I'm whining about this so much is because I think
> > tag should be something abstracted inside sysfs proper. It's something
> > which affects very internal operation of sysfs and I really want to keep
> > the implementation details inside sysfs. Spreading implementation over
> > kobject and sysfs didn't turn out too pretty after all.
>
> I agree. Most of the implementation is in sysfs already. We just have
> a few corner cases.
>
> Fundamentally it is the subsystems responsibility that creates the
> kobjects and the sysfs entries. The only case where I can see an
> ida generated number being a help is if we start having lifetime
> issues. Further the extra work to allocate and free tags ida based
> tags seems unnecessary.
>
> I don't doubt that there is a lot we can do better. My current goal
> is for something that is clean enough it won't get us into trouble
> later, and then merging the code. In tree where people can see
> the code and the interactions I expect it will be easier to talk
> about.
>
> Currently the interface with the users is very small. Adding the
> tag_type enumeration should make it smaller and make things more
> obviously static.
>
> Guys can we please make something useful happen?
>
> Eric

Now that the iproute2 patch is upstream, this patchset really is the
only thing keeping us from using network namespaces. Given that the

details of the tagging are trivially changeable with no abi changes, I'd
personally much rather see the patches go in as is, with whatever new
tagging patches Benjamin whips up, using ida or some new idea, being
applied later if we feel the need.

-serge

____

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

---

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by Tejun Heo on Tue, 01 Jul 2008 06:47:48 GMT

View Forum Message <> Reply to Message

Hello, Eric.

Eric W. Biederman wrote:
```
>> It's still dynamic from sysfs's POV and I think that will make
>> maintenance more difficult.
>
> Potentially.  I have no problem make it clear that things are more static.
```

Great. :-)

```
>> What you described is pretty much what I'm talking about.  The only
>> difference is whether to use caller-provided pointer as tag or an
>> ida-allocated integer.  The last sentence of the above paragraph is
>> basically sys_tag_enabled() function (maybe misnamed).
>
> So some concrete code examples here.  In the current code in lookup
> what I am doing is:
>
>  tag = sysfs_lookup_tag(parent_sd, parent->d_sb);
>  sd = sysfs_find_dirent(parent_sd, tag, dentry->d_name.name);
>
> With the proposed change of adding tag types sysfs_lookup_tag becomes:
>
> 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 = sysfs_info(sb)->tag[dir_sd->tag_type];
>
>  return tag;
> }
```

>
> Which means that in practice I can lookup that tag that I am displaying
> once.
>
> Then in sysfs_find_dirent we do:
>

```
>  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;
>  }
```

>
> That should keep the implementation sufficiently inside of sysfs for there
> to be no guessing. In addition as a practical matter we can only allow
> one tag to be visible in a directory at once or else we can not check
> for duplicate names. Which is the problem I see with a bitmap based test
> too unnecessary many degrees of freedom.

Having enumed tag types limits that a sb can have map to only one tag
but it doesn't really prevent multiple possibly visible entries which is
the real unnecessary degrees of freedom. That said, I don't really
think it's an issue.

> The number of tag types will be low as it is the number of subsystems
> that use the feature. Simple enough that I expect statically allocating
> the tag types in an enumeration is a safe and sane way to operate.
> i.e.
>

```
> enum sysfs_tag_types {
>  SYSFS_TAG_NETNS,
>  SYSFS_TAG_USERNS,
>        SYSFS_TAG_MAX
> };
```

I still would prefer something which is more generic. The abstraction
is clearer too. A sb shows untagged and a set of tags. A sd can either
be untagged or tagged (a single tag).

>> The main reason why I'm whining about this so much is because I think
>> tag should be something abstracted inside sysfs proper. It's something
>> which affects very internal operation of sysfs and I really want to keep
>> the implementation details inside sysfs. Spreading implementation over
>> kobject and sysfs didn't turn out too pretty after all.
>
> I agree. Most of the implementation is in sysfs already. We just have
> a few corner cases.

>
> Fundamentally it is the subsystems responsibility that creates the
> kobjects and the sysfs entries. The only case where I can see an
> ida generated number being a help is if we start having lifetime
> issues. Further the extra work to allocate and free tags ida based
> tags seems unnecessary.
>
> I don't doubt that there is a lot we can do better. My current goal
> is for something that is clean enough it won't get us into trouble
> later, and then merging the code. In tree where people can see
> the code and the interactions I expect it will be easier to talk
> about.
>
> Currently the interface with the users is very small. Adding the
> tag_type enumeration should make it smaller and make things more
> obviously static.

Using ida (or idr if a pointer for private data is necessary) is really
easy. It'll probably take a few tens of lines of code. That said, I
don't think I have enough rationale to nack what you described. So, as
long as the tags are made static, I won't object.

Thanks.

--
tejun

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

---

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by ebiederm on Tue, 01 Jul 2008 07:50:04 GMT

View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Now that the iproute2 patch is upstream, this patchset really is the
> only thing keeping us from using network namespaces. Given that the
> details of the tagging are trivially changeable with no abi changes, I'd
> personally much rather see the patches go in as is, with whatever new
> tagging patches Benjamin whips up, using ida or some new idea, being
> applied later if we feel the need.

My point exactly. No one seems to contest the userspace semantics so
as long as we don't put ourselves into a real mess we should be fine.

Eric

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

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by ebiederm on Tue, 01 Jul 2008 09:20:20 GMT

View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> Hello, Eric.
>
> Eric W. Biederman wrote:
>>> It's still dynamic from sysfs's POV and I think that will make
>>> maintenance more difficult.
>>
>> Potentially. I have no problem make it clear that things are more static.
>
> Great. :-)
>

> Having enumed tag types limits that a sb can have map to only one tag
> but it doesn't really prevent multiple possibly visible entries which is
> the real unnecessary degrees of freedom. That said, I don't really
> think it's an issue.

Having a single tag type per directory and thus a single tag visible per
directory does prevent multiple possible visible entries.

That is we can check when we add the sd if there will be a conflict in
the directory.

>> The number of tag types will be low as it is the number of subsystems
>> that use the feature. Simple enough that I expect statically allocating
>> the tag types in an enumeration is a safe and sane way to operate.
>> i.e.
>>
>> enum sysfs_tag_types {
>> SYSFS_TAG_NETNS,
>> SYSFS_TAG_USERNS,
>>   SYSFS_TAG_MAX
>> };
>
> I still would prefer something which is more generic. The abstraction
> is clearer too. A sb shows untagged and a set of tags. A sd can either

> be untagged or tagged (a single tag).

That is the abstraction now.

The only difference is how we represent the set of tags.
I use and array of the valid tags.
You use a bitmap.

And array allows the lookup of the tag I am looking for before I search for the sd. An bitmap requires me to compare each entry.

For me that is a deal breaker. Currently in certain pathological cases we have scaling issues with sysctl and sysfs that we can have enormous directories that start running slowly. To fix lookup performance requires that we know the full name before we do the directory search which is the name string and the tag.

So I having a type of tag as being of fundamental importance in the interface now so we don't need to refactor all of the users later. In addition to the fact that we need the type to know how to set the tags when mounting a superblock and when given a new kobject to create an sd for.

We could make the types dynamic rather then a static enumeration but that seems needless complexity for now.

> Using ida (or idr if a pointer for private data is necessary) is really
> easy. It'll probably take a few tens of lines of code. That said, I
> don't think I have enough rationale to nack what you described. So, as
> long as the tags are made static, I won't object.

Sounds good. The only justification I can think of for ida tags is that they are smaller, and so can keep the sysfs_dirents smaller. Which occasionally is a significant concern. Still that should be an optimization that we can apply later, as it is not a structural difference in the code.

Just to confirm. Do you the two operations:
  mount_tag - called only when the sb is mounted
  kobject_tag - called when we create new sd or rename an sd

Cause you to view an the tags as dynamic?

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

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by Tejun Heo on Tue, 01 Jul 2008 10:30:41 GMT

View Forum Message <> Reply to Message

Hello,

Eric W. Biederman wrote:
>> Having enumed tag types limits that a sb can have map to only one tag
>> but it doesn't really prevent multiple possibly visible entries which is
>> the real unnecessary degrees of freedom. That said, I don't really
>> think it's an issue.
>
> Having a single tag type per directory and thus a single tag visible per
> directory does prevent multiple possible visible entries.
>
> That is we can check when we add the sd if there will be a conflict in
> the directory.

Yeap, that we can do.

>> I still would prefer something which is more generic. The abstraction
>> is clearer too. A sb shows untagged and a set of tags. A sd can either
>> be untagged or tagged (a single tag).
>
> That is the abstraction now.
>
> The only difference is how we represent the set of tags.
> I use and array of the valid tags.
> You use a bitmap.
>
> And array allows the lookup of the tag I am looking for before
> I search for the sd. An bitmap requires me to compare each entry.

How so? sysfs_sb->bitmap which contains enough bits for all the defined
tags and determining whether a sd should be shown or not is as simple as
single test_bit.

> For me that is a deal breaker. Currently in certain pathological
> cases we have scaling issues with sysctl and sysfs that we can
> have enormous directories that start running slowly. To fix
> lookup performance requires that we know the full name before
> we do the directory search which is the name string and the
> tag.
>
> So I having a type of tag as being of fundamental importance in
> the interface now so we don't need to refactor all of the users
> later. In addition to the fact that we need the type to know
> how to set the tags when mounting a superblock and when
> given a new kobject to create an sd for.

>
> We could make the types dynamic rather then a static enumeration but
> that seems needless complexity for now.

What I'm feeling unease about is the extra level of abstraction added by
tag types. A sd is given a tag. A sb shows a set of tags. The most
straight forward to implement that is to give sd a tag and test the tag
against sb's set of tags. The type is added because pointer tag
requires sequential matching which is usually best to avoid. It's
nothing fundamental. It's an extra baggage.

>> Using ida (or idr if a pointer for private data is necessary) is really
>> easy. It'll probably take a few tens of lines of code. That said, I
>> don't think I have enough rationale to nack what you described. So, as
>> long as the tags are made static, I won't object.
>
> Sounds good. The only justification I can think of for ida tags is that
> they are smaller, and so can keep the sysfs_dirents smaller.   Which
> occasionally is a significant concern. Still that should be an optimization
> that we can apply later, as it is not a structural difference in the code.
>
> Just to confirm. Do you the two operations:
>   mount_tag - called only when the sb is mounted
>   kobject_tag - called when we create new sd or rename an sd
>
> Cause you to view an the tags as dynamic?

The thing is that I don't really see why there's tagged_dir_ops at all.
What's needed is tagged sd's and sb's which can show subset of those
tags, so adding callback ops for tags just doesn't make much sense to
me. The interface should ideally be...

1. alloc/release tag
2. set / change / remove tag on sd
3. enable / disable tag on a sb

This has been my opinion from the beginning. Unless the tags need to be
changed dynamically on demand (which I hope is not the case), there just
is plainly no reason to have callbacks for tags.

Thanks.

--
tejun

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by ebiederm on Tue, 01 Jul 2008 12:30:47 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> Hello,
>
> Eric W. Biederman wrote:
>>> Having enumed tag types limits that a sb can have map to only one tag
>>> but it doesn't really prevent multiple possibly visible entries which is
>>> the real unnecessary degrees of freedom. That said, I don't really
>>> think it's an issue.
>>
>> Having a single tag type per directory and thus a single tag visible per
>> directory does prevent multiple possible visible entries.
>>
>> That is we can check when we add the sd if there will be a conflict in
>> the directory.
>
> Yeap, that we can do.

What we are implementing is not, a sb with a set of tags that are displayed,
but directories with a single tag that is displayed. The sb just happens
to hold the state for the directories.

A directory displaying only a single tag is an necessary constraint for
a large number of reasons.

>> And array allows the lookup of the tag I am looking for before
>> I search for the sd. An bitmap requires me to compare each entry.
>
> How so? sysfs_sb->bitmap which contains enough bits for all the defined
> tags and determining whether a sd should be shown or not is as simple as
> single test_bit.

Yes. The compare happens to be test_bit.

With a bitmap you must visit each dirent with a given name and see if
it has a tag that is displayed.

With an array you can lookup the tag aprori and can potentially do a
hash table lookup or a tree lookup and are not required to visit each
entry.

> What I'm feeling unease about is the extra level of abstraction added by
> tag types. A sd is given a tag. A sb shows a set of tags. The most
> straight forward to implement that is to give sd a tag and test the tag
> against sb's set of tags. The type is added because pointer tag

> requires sequential matching which is usually best to avoid. It's
> nothing fundamental. It's an extra baggage.

That is just one important aspect of it.   We need a way to describe
which tag a sb,directory pair displays. It is a fundamental concept.

>>> Using ida (or idr if a pointer for private data is necessary) is really
>>> easy. It'll probably take a few tens of lines of code. That said, I
>>> don't think I have enough rationale to nack what you described. So, as
>>> long as the tags are made static, I won't object.
>>
>> Sounds good. The only justification I can think of for ida tags is that
>> they are smaller, and so can keep the sysfs_dirents smaller.   Which
>> occasionally is a significant concern. Still that should be an optimization
>> that we can apply later, as it is not a structural difference in the code.
>>
>> Just to confirm. Do you the two operations:
>>   mount_tag - called only when the sb is mounted
>>   kobject_tag - called when we create new sd or rename an sd
>>
>> Cause you to view an the tags as dynamic?
>
> The thing is that I don't really see why there's tagged_dir_ops at all.

We need callbacks for interfacing with the kobject layer, and for
selecting our set of tags at mount time. Not tagged_dir_ops so much
as tagged_type_ops.

> What's needed is tagged sd's and sb's which can show subset of those
> tags, so adding callback ops for tags just doesn't make much sense to
> me. The interface should ideally be...

> 1. alloc/release tag
    Agreed.

> 2. set / change / remove tag on sd
    Essentially agreed.

    Create an sd with a tag, change the tag on a sd.
    Having an untagged sd in a directory that requires tags should
    not be allowed.

> 3. enable / disable tag on a sb
    Disagree that is too flexible. Tags on a sb need to be
    unchanging or else we get vfs layer issues.

   Further the abstraction is logically exactly one tag on a
   (sb,directory) pair.

    The operations needed are.
    - Select the set of tags on a sb (at mount time)
      This requires we call a set of callbacks. [My mount_sb callback]

    - release a tag (which implies removing all tagged entries and
      removing the sb reference)

4. Interface with the kobject layer.
   kobject_add calls sysfs_create_dir
   kboject_rename calls sysfs_rename_dir
   kobject_del calls sysfs_remove_dir

   For the first two operations we need a helper function to go from a
   kobject to a tag.

   For the second two operations we need to go from a kobject to a sd.

> This has been my opinion from the beginning. Unless the tags need to be
> changed dynamically on demand (which I hope is not the case), there just
> is plainly no reason to have callbacks for tags.

We don't need callbacks to poll to see if the tags on a sd have
changed.

We need helper functions for interfacing with the rest of the kernel.

Eric

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

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by Tejun Heo on Wed, 02 Jul 2008 03:24:42 GMT

View Forum Message <> Reply to Message

Hello,

Eric W. Biederman wrote:
> What we are implementing is not, a sb with a set of tags that are displayed,
> but directories with a single tag that is displayed. The sb just happens
> to hold the state for the directories.
>
> A directory displaying only a single tag is an necessary constraint for
> a large number of reasons.

Okay, that isn't exactly the impression I get but... well. Let's see.

>>> And array allows the lookup of the tag I am looking for before
>>> I search for the sd. An bitmap requires me to compare each entry.
>> How so? sysfs_sb->bitmap which contains enough bits for all the defined
>> tags and determining whether a sd should be shown or not is as simple as
>> single test_bit.
>
> Yes. The compare happens to be test_bit.
>
> With a bitmap you must visit each dirent with a given name and see if
> it has a tag that is displayed.
>
> With an array you can lookup the tag aprori and can potentially do a
> hash table lookup or a tree lookup and are not required to visit each
> entry.

A few things...

1. The lookup is currently done linearly and is fast enough for now.
Also, most lookup ops are cached by vfs layer. I'm not sure how
probable it is that we're gonna need hash or tree based sd lookup.

2. I don't think it's gonna be too difficult to speed up bitmap based
lookup. It would require a bit more intelligence but there's no
fundamental restriction. Just organizing the tree by tag first would
give us the same order of magnitude lookup given that the tags are used
the same way.

>> What I'm feeling unease about is the extra level of abstraction added by
>> tag types. A sd is given a tag. A sb shows a set of tags. The most
>> straight forward to implement that is to give sd a tag and test the tag
>> against sb's set of tags. The type is added because pointer tag
>> requires sequential matching which is usually best to avoid. It's
>> nothing fundamental. It's an extra baggage.
>
> That is just one important aspect of it.   We need a way to describe
> which tag a sb,directory pair displays. It is a fundamental concept.

For netns, yes. I just think it would be better if the sysfs mechanism
to support that concept is more generic especially because it doesn't
seem too difficult to make it that way.

>>> Cause you to view an the tags as dynamic?
>> The thing is that I don't really see why there's tagged_dir_ops at all.
>
> We need callbacks for interfacing with the kobject layer, and for
> selecting our set of tags at mount time. Not tagged_dir_ops so much

> as tagged_type_ops.

The kobject op seems a bit strange way to interface to me. For mount,
yeah, we'll need a hook somewhere or pass it via mount option maybe.

>> What's needed is tagged sd's and sb's which can show subset of those
>> tags, so adding callback ops for tags just doesn't make much sense to
>> me. The interface should ideally be...
>
>> 1. alloc/release tag
>   Agreed.
>
>> 2. set / change / remove tag on sd
>   Essentially agreed.
>
>   Create an sd with a tag, change the tag on a sd.
>   Having an untagged sd in a directory that requires tags should
>   not be allowed.
>
>> 3. enable / disable tag on a sb
>   Disagree that is too flexible. Tags on a sb need to be
>   unchanging or else we get vfs layer issues.

Yeah, this really should be something which can't change once it's mounted.

>   Further the abstraction is logically exactly one tag on a
>   (sb,directory) pair.

I'm not so sure here. As a policy, maybe but I don't really see a
fundamental reason that the mechanism should enforce this.

>   The operations needed are.
>   - Select the set of tags on a sb (at mount time)
>   This requires we call a set of callbacks. [My mount_sb callback]
>
>   - release a tag (which implies removing all tagged entries and
>   removing the sb reference)
>
> 4. Interface with the kobject layer.
>   kobject_add calls sysfs_create_dir
>   kboject_rename calls sysfs_rename_dir
>   kobject_del calls sysfs_remove_dir
>
>   For the first two operations we need a helper function to go from a
>   kobject to a tag.

Why not just add a parameter to sysfs_create_dir()? It's just twisted.

>   For the second two operations we need to go from a kobject to a sd.
>
>> This has been my opinion from the beginning. Unless the tags need to be
>> changed dynamically on demand (which I hope is not the case), there just
>> is plainly no reason to have callbacks for tags.
>
> We don't need callbacks to poll to see if the tags on a sd have
> changed.
>
> We need helper functions for interfacing with the rest of the kernel.

Yes, that's why I view it as strange. These can be done in forward way
(by passing in mount options and/or arguments) but it's done by first
going into the sysfs and then calling back out to outer layer.

Thanks.

--
tejun

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

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by ebiederm on Wed, 02 Jul 2008 03:53:00 GMT

View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> Hello,
>
> Eric W. Biederman wrote:
>> What we are implementing is not, a sb with a set of tags that are displayed,
>> but directories with a single tag that is displayed. The sb just happens
>> to hold the state for the directories.
>>
>> A directory displaying only a single tag is an necessary constraint for
>> a large number of reasons.
>
> Okay, that isn't exactly the impression I get but... well. Let's see.

Well one of those reasons is not having duplicate entries in your directory listing.
That is much harder otherwise.

> A few things...
>

> 1. The lookup is currently done linearly and is fast enough for now.
> Also, most lookup ops are cached by vfs layer. I'm not sure how
> probable it is that we're gonna need hash or tree based sd lookup.

I don't know how bad sysfs is. On the sysctl side I have people complaining
because I am doing a lookup during insert and that lookup is linear. Sysfs
appears to have the same complexity as sysctl but just smaller constants.

>> That is just one important aspect of it.   We need a way to describe
>> which tag a sb,directory pair displays. It is a fundamental concept.
>
> For netns, yes. I just think it would be better if the sysfs mechanism
> to support that concept is more generic especially because it doesn't
> seem too difficult to make it that way.

Well the envisioned use is for other namespaces and they all are similar
to the network namespace in that way.

>>>> Cause you to view an the tags as dynamic?
>>> The thing is that I don't really see why there's tagged_dir_ops at all.
>>
>> We need callbacks for interfacing with the kobject layer, and for
>> selecting our set of tags at mount time. Not tagged_dir_ops so much
>> as tagged_type_ops.
>
> The kobject op seems a bit strange way to interface to me. For mount,
> yeah, we'll need a hook somewhere or pass it via mount option maybe.

I will look how if there is a place in the kobject layer to put it. With
a second but noticeably different user I can compare and see how hard that will be.

>>> 3. enable / disable tag on a sb
>>   Disagree that is too flexible. Tags on a sb need to be
>>   unchanging or else we get vfs layer issues.
>
> Yeah, this really should be something which can't change once it's mounted.
The VFS chokes otherwise because it can't cache things properly.

>>   Further the abstraction is logically exactly one tag on a
>>   (sb,directory) pair.
>
> I'm not so sure here. As a policy, maybe but I don't really see a
> fundamental reason that the mechanism should enforce this.

Well in the first implementation.

>> 4. Interface with the kobject layer.
>>   kobject_add calls sysfs_create_dir

>>   kboject_rename calls sysfs_rename_dir
>>   kobject_del calls sysfs_remove_dir
>>
>>   For the first two operations we need a helper function to go from a
>>   kobject to a tag.
>
> Why not just add a parameter to sysfs_create_dir()? It's just twisted.

I added it where it was easiest. Adding a parameter to sysfs_create_dir
simply means I have to add the function to the kobject layer. It is certainly
worth a second look though.

>> We need helper functions for interfacing with the rest of the kernel.
>
> Yes, that's why I view it as strange. These can be done in forward way
> (by passing in mount options and/or arguments) but it's done by first
> going into the sysfs and then calling back out to outer layer.

Well in the case of mount the default parameter at least is current, and
there are good reasons for that.

On the other side I can't pass a tag through from the device layer to
the kobject layer. It isn't a concept the kobject layer supports.

At least though the conversation is in relative agreement. I will refresh
the patches shortly and see where we are at.

Eric

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by Tejun Heo on Wed, 02 Jul 2008 04:37:32 GMT

View Forum Message <> Reply to Message

Hello,

Eric W. Biederman wrote:
>>> A directory displaying only a single tag is an necessary constraint for
>>> a large number of reasons.
>> Okay, that isn't exactly the impression I get but... well. Let's see.
>
> Well one of those reasons is not having duplicate entries in your directory listing.
> That is much harder otherwise.

Agreed.

>> For netns, yes. I just think it would be better if the sysfs mechanism
>> to support that concept is more generic especially because it doesn't
>> seem too difficult to make it that way.
>
> Well the envisioned use is for other namespaces and they all are similar
> to the network namespace in that way.

Something I've been curious about is a directory which contains both the
untagged entries and tagged ones. I can definitely imagine something
like that to be useful for block device namespace.

>>>>> Cause you to view an the tags as dynamic?
>>>> The thing is that I don't really see why there's tagged_dir_ops at all.
>>> We need callbacks for interfacing with the kobject layer, and for
>>> selecting our set of tags at mount time. Not tagged_dir_ops so much
>>> as tagged_type_ops.
>> The kobject op seems a bit strange way to interface to me. For mount,
>> yeah, we'll need a hook somewhere or pass it via mount option maybe.
>
> I will look how if there is a place in the kobject layer to put it. With
> a second but noticeably different user I can compare and see how hard that will be.

Great, thanks.

>>>   Further the abstraction is logically exactly one tag on a
>>>   (sb,directory) pair.
>> I'm not so sure here. As a policy, maybe but I don't really see a
>> fundamental reason that the mechanism should enforce this.
>
> Well in the first implementation.

This pretty much defines the interface and is likely to force future
users to fit themselves into it.

>>> 4. Interface with the kobject layer.
>>>   kobject_add calls sysfs_create_dir
>>>   kboject_rename calls sysfs_rename_dir
>>>   kobject_del calls sysfs_remove_dir
>>>
>>>   For the first two operations we need a helper function to go from a
>>>   kobject to a tag.
>> Why not just add a parameter to sysfs_create_dir()? It's just twisted.
>
> I added it where it was easiest. Adding a parameter to sysfs_create_dir
> simply means I have to add the function to the kobject layer. It is certainly

> worth a second look though.

Is it difficult to just export it via kobject and device layer? If
changing the default function is too much of a hassle (and I'm sure it
would be), just add an extended version which takes @tag. The current
implementation feels like it tried too hard to not add intermediate
interfaces and ended up shooting outside from the innermost layer.

>>> We need helper functions for interfacing with the rest of the kernel.
>> Yes, that's why I view it as strange. These can be done in forward way
>> (by passing in mount options and/or arguments) but it's done by first
>> going into the sysfs and then calling back out to outer layer.
>
> Well in the case of mount the default parameter at least is current, and
> there are good reasons for that.

I was imagining something like...

 mount -t sysfs -o ns=0,4,5 /my/sys

And let the userland control which ns's are visible in the particular
mount. I'm not sure how useful that will be tho.

> On the other side I can't pass a tag through from the device layer to
> the kobject layer. It isn't a concept the kobject layer supports.

I think it's best to make kobject layer support it.

> At least though the conversation is in relative agreement. I will refresh
> the patches shortly and see where we are at.

Thanks a lot for the patience. :-)

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

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by Andreas B Aaen on Wed, 02 Jul 2008 07:18:53 GMT
View Forum Message <> Reply to Message

On Wednesday 02 July 2008 06:37, Tejun Heo wrote:
> I was imagining something like...
>

> mount -t sysfs -o ns=0,4,5 /my/sys
>
> And let the userland control which ns's are visible in the particular
> mount. I'm not sure how useful that will be tho.

Useful. However I didn't know that an indexnumber was associated with the network namespaces.

Especially if you want to use network namespaces for scaleability and not isolation purposes.

Regards,
--
Andreas Bach Aaen   System Developer, M. Sc.
Tieto Enator A/S   tel: +45 89 38 51 00
Skanderborgvej 232   fax: +45 89 38 51 01
8260 Viby J   Denmark   andreas.aaen@tietoenator.com
____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by ebiederm on Wed, 02 Jul 2008 16:49:33 GMT

View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> Is it difficult to just export it via kobject and device layer?

Well gregkh thought it wasn't a good idea last time I tried exploring that.

> If
> changing the default function is too much of a hassle (and I'm sure it
> would be), just add an extended version which takes @tag. The current
> implementation feels like it tried too hard to not add intermediate
> interfaces and ended up shooting outside from the innermost layer.

It tried for something that was simple to use and that worked.

Also the way things work. I have to use all of the intermediate layers and their calls to various functions. So just passing a parameter through doesn't work to well.

It looks to me like the clean solution is move kobject_tag into kobj_type, and have it call some higher level function.

We also need to remove the maintenance disaster that is
kobject_set_name from sysfs_rename_dir. And push it into
kobject_rename instead. The error handling is harder in
that case but otherwise we should be in good shape.

>> On the other side I can't pass a tag through from the device layer to
>> the kobject layer. It isn't a concept the kobject layer supports.
>
> I think it's best to make kobject layer support it.

Assuming Greg will accept it when he sees reasonable patches.

Eric

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

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by gregkh on Thu, 03 Jul 2008 00:15:49 GMT

View Forum Message <> Reply to Message

On Wed, Jul 02, 2008 at 09:49:33AM -0700, Eric W. Biederman wrote:
> Assuming Greg will accept it when he sees reasonable patches.

I always accept "reasonable patches" :)

thanks,

greg k-h

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

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by Tejun Heo on Thu, 03 Jul 2008 03:18:45 GMT

View Forum Message <> Reply to Message

Hello, Eric.

Eric W. Biederman wrote:
>> If
>> changing the default function is too much of a hassle (and I'm sure it

>> would be), just add an extended version which takes @tag. The current
>> implementation feels like it tried too hard to not add intermediate
>> interfaces and ended up shooting outside from the innermost layer.
>
> It tried for something that was simple to use and that worked.
>
> Also the way things work. I have to use all of the intermediate layers
> and their calls to various functions. So just passing a parameter through
> doesn't work to well.

There is rather large possibility that I'm just being dumb here
especially because I haven't reviewed the users of this facility, so all
the comments I'm making are from the POV of interfaces of sysfs and the
related layers. I think I've made my concerns clear by now. If you
still think the callbacks are the best way to go, please try to
enlighten me. I really don't wanna be stopping something which is
better from ignorance. Just give me some concrete examples or point me
to codes which show how and why the current interface is the best for
the users and switching isn't a good idea.

> It looks to me like the clean solution is move kobject_tag into
> kobj_type, and have it call some higher level function.
>
> We also need to remove the maintenance disaster that is
> kobject_set_name from sysfs_rename_dir. And push it into
> kobject_rename instead. The error handling is harder in
> that case but otherwise we should be in good shape.

Heh... I personally think kobject layer as a whole should just be hidden
under the cabinet of device driver model but I'm having difficult time
convincing other people of it. Anyways, fully agree the interaction
between kobject and sysfs is ugly at a lot of places.

>>> On the other side I can't pass a tag through from the device layer to
>>> the kobject layer. It isn't a concept the kobject layer supports.
>> I think it's best to make kobject layer support it.
>
> Assuming Greg will accept it when he sees reasonable patches.

Greg says he would. :-)

Thanks a lot for your patience.

--
tejun
____
Containers mailing list
Containers@lists.linux-foundation.org

https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by ebiederm on Thu, 03 Jul 2008 05:11:08 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> There is rather large possibility that I'm just being dumb here
> especially because I haven't reviewed the users of this facility, so all
> the comments I'm making are from the POV of interfaces of sysfs and the
> related layers. I think I've made my concerns clear by now. If you
> still think the callbacks are the best way to go, please try to
> enlighten me. I really don't wanna be stopping something which is
> better from ignorance. Just give me some concrete examples or point me
> to codes which show how and why the current interface is the best for
> the users and switching isn't a good idea.

Currently I think a callback on to get the tag from a kobject is the
best way to go. That way we don't need to add a field to struct
kobject (and don't need the associated redundancy), and we can lookup
up the tag when we need it.

I have been playing with the code and just about have it ready
to go. I just need to refactor all of my changes into clean
patches at this point, plus a bit of review and test. Ben & Daniel
have given me a version of the previous patchset rebased unto the
latest -mm so that should help for the unchanged parts.

Introducing the sysfs_tag_type thing and pushing the functions to
the edges helps. It especially cleans up the ugly mount/umount
situation allowing us to handle that with generic code.

Moving the kobject_tag into struct ktype works and looks roughly
as clean as what happens with attributes. So I that seems reasonable,
and doesn't result in a significant change in the users.

The result of which means that I only have the helper function sysfs_creation_tag
left in sysfs/dir.c Left in there are some of the nasties in dealing with symlinks.

At this point I believe I have achieved a nice degree of simplifying the sysfs
code in the current patches without really changing the users or
making it more complex for them.

I have not implemented ida tags, and I don't plan to. That is just
unnecessary work right now. The users are simple and the meat of the
logic would not change so it should be simple to add.

>> It looks to me like the clean solution is move kobject_tag into
>> kobj_type, and have it call some higher level function.
>>
>> We also need to remove the maintenance disaster that is
>> kobject_set_name from sysfs_rename_dir. And push it into
>> kobject_rename instead. The error handling is harder in
>> that case but otherwise we should be in good shape.
>
> Heh... I personally think kobject layer as a whole should just be hidden
> under the cabinet of device driver model but I'm having difficult time
> convincing other people of it. Anyways, fully agree the interaction
> between kobject and sysfs is ugly at a lot of places.

I would be happy if we could remove all nonsense kobject that are there just
for structural purposes but have no purpose otherwise. Things like kobjects
for symlinks. The kobject layer doesn't seem to have a clear identity
and purpose that I can see right now.

> Thanks a lot for your patience.

Welcome. The code reached a point a while ago where it didn't make sense
to change it without review feedback.

Eric

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

---

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by Daniel Lezcano on Thu, 03 Jul 2008 10:56:17 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Tejun Heo <htejun@gmail.com> writes:
>
>> There is rather large possibility that I'm just being dumb here
>> especially because I haven't reviewed the users of this facility, so all
>> the comments I'm making are from the POV of interfaces of sysfs and the
>> related layers. I think I've made my concerns clear by now. If you
>> still think the callbacks are the best way to go, please try to
>> enlighten me. I really don't wanna be stopping something which is
>> better from ignorance. Just give me some concrete examples or point me
>> to codes which show how and why the current interface is the best for
>> the users and switching isn't a good idea.

>
> Currently I think a callback on to get the tag from a kobject is the
> best way to go. That way we don't need to add a field to struct
> kobject (and don't need the associated redundancy), and we can lookup
> up the tag when we need it.

The kobject events are sent through a netlink message which is not
currently per network namespace. Shouldn't be useful to have a way to
retrieve from the kobject the network namespace or the uevent socket
associated with it ? IMHO having idr in the kobject + netns pointer
associated may help to handle the sysfs isolation and makes the uevent
per namespace trivial, no ?

> I have been playing with the code and just about have it ready
> to go. I just need to refactor all of my changes into clean
> patches at this point, plus a bit of review and test. Ben & Daniel
> have given me a version of the previous patchset rebased unto the
> latest -mm so that should help for the unchanged parts.
>
> Introducing the sysfs_tag_type thing and pushing the functions to
> the edges helps. It especially cleans up the ugly mount/umount
> situation allowing us to handle that with generic code.
>
> Moving the kobject_tag into struct ktype works and looks roughly
> as clean as what happens with attributes. So I that seems reasonable,
> and doesn't result in a significant change in the users.
>
> The result of which means that I only have the helper function sysfs_creation_tag
> left in sysfs/dir.c Left in there are some of the nasties in dealing with symlinks.
>
> At this point I believe I have achieved a nice degree of simplifying the sysfs
> code in the current patches without really changing the users or
> making it more complex for them.
>
> I have not implemented ida tags, and I don't plan to. That is just
> unnecessary work right now. The users are simple and the meat of the
> logic would not change so it should be simple to add.
>
>>> It looks to me like the clean solution is move kobject_tag into
>>> kobj_type, and have it call some higher level function.
>>>
>>> We also need to remove the maintenance disaster that is
>>> kobject_set_name from sysfs_rename_dir. And push it into
>>> kobject_rename instead. The error handling is harder in
>>> that case but otherwise we should be in good shape.
>> Heh... I personally think kobject layer as a whole should just be hidden
>> under the cabinet of device driver model but I'm having difficult time
>> convincing other people of it. Anyways, fully agree the interaction

>> between kobject and sysfs is ugly at a lot of places.
>
> I would be happy if we could remove all nonsense kobject that are there just
> for structural purposes but have no purpose otherwise. Things like kobjects
> for symlinks. The kobject layer doesn't seem to have a clear identity
> and purpose that I can see right now.
>
>> Thanks a lot for your patience.
>
> Welcome. The code reached a point a while ago where it didn't make sense
> to change it without review feedback.
>
> Eric
>
> ____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers
>

--

Sauf indication contraire ci-dessus:
Compagnie IBM France
Si
Courbevoie
RCS Nanterre 552 118 465
Forme Sociale : S.A.S.
Capital Social : 542.737.118 ?
SIREN/SIRET : 552 118 465 02430

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

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by ebiederm on Thu, 03 Jul 2008 12:27:01 GMT

View Forum Message <> Reply to Message

Daniel Lezcano <dlezcano@fr.ibm.com> writes:

> The kobject events are sent through a netlink message which is not currently per
> network namespace. Shouldn't be useful to have a way to retrieve from the
> kobject the network namespace or the uevent socket associated with it ? IMHO
> having idr in the kobject + netns pointer associated may help to handle the
> sysfs isolation and makes the uevent per namespace trivial, no ?

Grumble. I have been conveniently been forgetting about that socket.
Similarly we have the user mode helpers to deal with.

For this conversation there is a simple answer. All of that is in the
kobject layer, and works even when you compile sysfs out of your kernel.
Therefore it is a separate problem. And sysfs idr tags have nothing
to do with it.

It is most definitely something we need to come back to. I bet there
are some interesting interactions when you have multiple network devices
with the same name generating events.

Eric

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

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by Benjamin Thery on Thu, 03 Jul 2008 12:37:49 GMT

View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Daniel Lezcano <dlezcano@fr.ibm.com> writes:
>
>> The kobject events are sent through a netlink message which is not currently per
>> network namespace. Shouldn't be useful to have a way to retrieve from the
>> kobject the network namespace or the uevent socket associated with it ? IMHO
>> having idr in the kobject + netns pointer associated may help to handle the
>> sysfs isolation and makes the uevent per namespace trivial, no ?
>
> Grumble. I have been conveniently been forgetting about that socket.
> Similarly we have the user mode helpers to deal with.

>
> For this conversation there is a simple answer. All of that is in the
> kobject layer, and works even when you compile sysfs out of your kernel.
> Therefore it is a separate problem. And sysfs idr tags have nothing
> to do with it.

> It is most definitely something we need to come back to. I bet there
> are some interesting interactions when you have multiple network devices
> with the same name generating events.

Indeed, we observed some fun things with one distro (which defines some particular udev rules) when a device called eth0 in a namespace comes back to init net :)

Benjamin

>
> Eric
>
>
>
>
>
>
>
>
>
>

--
Benjamin Thery - BULL/DT/Open Software R&D

   http://www.bull.com

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

---

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by Daniel Lezcano on Thu, 03 Jul 2008 12:55:38 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Daniel Lezcano <dlezcano@fr.ibm.com> writes:
>
>> The kobject events are sent through a netlink message which is not currently per

>> network namespace. Shouldn't be useful to have a way to retrieve from the
>> kobject the network namespace or the uevent socket associated with it ? IMHO
>> having idr in the kobject + netns pointer associated may help to handle the
>> sysfs isolation and makes the uevent per namespace trivial, no ?
>
> Grumble. I have been conveniently been forgetting about that socket.
> Similarly we have the user mode helpers to deal with.
>
> For this conversation there is a simple answer. All of that is in the
> kobject layer, and works even when you compile sysfs out of your kernel.
> Therefore it is a separate problem. And sysfs idr tags have nothing
> to do with it.

Ah Ok, I am not really familiar with kobject/sysfs so I thought there
was a proposition to store the id in the kobject instead of using the
tag callbacks, so I figured, perhaps, the idr could have been used in
the kobject layer and the sysfs being built upon that.

> It is most definitely something we need to come back to. I bet there
> are some interesting interactions when you have multiple network devices
> with the same name generating events.

Yes as mentionned Benjamin, we have the eth0 in the init_net which is
shut down when a network namespace with a netdev with the same name
exits. There is a udev rule which ifdown eth0 :)
____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by Tejun Heo on Thu, 03 Jul 2008 15:58:23 GMT

View Forum Message <> Reply to Message

Hello,

Eric W. Biederman wrote:
> Daniel Lezcano <dlezcano@fr.ibm.com> writes:
>
>> The kobject events are sent through a netlink message which is not currently per
>> network namespace. Shouldn't be useful to have a way to retrieve from the
>> kobject the network namespace or the uevent socket associated with it ? IMHO
>> having idr in the kobject + netns pointer associated may help to handle the
>> sysfs isolation and makes the uevent per namespace trivial, no ?
>
> Grumble. I have been conveniently been forgetting about that socket.
> Similarly we have the user mode helpers to deal with.

>
> For this conversation there is a simple answer. All of that is in the
> kobject layer, and works even when you compile sysfs out of your kernel.
> Therefore it is a separate problem. And sysfs idr tags have nothing
> to do with it.
>
> It is most definitely something we need to come back to. I bet there
> are some interesting interactions when you have multiple network devices
> with the same name generating events.

Related delta: I've been thinking that uevents should be part of sysfs
not kobject as that's what the userland is gonna associate the event
with. Would that solve the problem you're thinking about?

Thanks.

--
tejun

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

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by Daniel Lezcano on Thu, 03 Jul 2008 18:29:42 GMT

View Forum Message <> Reply to Message

Tejun Heo wrote:
> Hello,
>
> Eric W. Biederman wrote:
>> Daniel Lezcano <dlezcano@fr.ibm.com> writes:
>>
>>> The kobject events are sent through a netlink message which is not currently per
>>> network namespace. Shouldn't be useful to have a way to retrieve from the
>>> kobject the network namespace or the uevent socket associated with it ? IMHO
>>> having idr in the kobject + netns pointer associated may help to handle the
>>> sysfs isolation and makes the uevent per namespace trivial, no ?
>> Grumble. I have been conveniently been forgetting about that socket.
>> Similarly we have the user mode helpers to deal with.
>>
>> For this conversation there is a simple answer. All of that is in the
>> kobject layer, and works even when you compile sysfs out of your kernel.
>> Therefore it is a separate problem. And sysfs idr tags have nothing
>> to do with it.
>>
>> It is most definitely something we need to come back to. I bet there

>> are some interesting interactions when you have multiple network devices
>> with the same name generating events.
>
> Related delta: I've been thinking that uevents should be part of sysfs
> not kobject as that's what the userland is gonna associate the event
> with. Would that solve the problem you're thinking about?

uevents can work with the network namespaces being compiled in and the
sysfs compiled out. AFAICS, uevents will be unable to handle multiple
network namespaces if it is tied with sysfs, no ?

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

---

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by ebiederm on Thu, 03 Jul 2008 19:57:14 GMT
View Forum Message <> Reply to Message

Benjamin Thery <benjamin.thery@bull.net> writes:

> Indeed, we observed some fun things with one distro (which defines some
> particular udev rules) when a device called eth0 in a namespace comes back to
> init net :)

Speaking of. Don't let me forget but I have a patch I need to send out
that deletes pseudo devices instead of sending them back to eth0.   We can't
do that for real hardware obviously but for things like veth and macvlan
devices it greatly simplifies the cleanup.

Eric

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

---

Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by ebiederm on Thu, 03 Jul 2008 20:08:52 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> Related delta: I've been thinking that uevents should be part of sysfs
> not kobject as that's what the userland is gonna associate the event
> with. Would that solve the problem you're thinking about?

The good news is that uevent_sock is currently restricted to just
the initial network namespace (so the functionality completely
disappears in the other namespaces), and that it is broadcast only.

So it should be possible to look at who the client is and by some
magic criterian decide if it should receive the broadcast message.

The call to the user mode helper is trickier. How do we setup the
proper user space context.

None of this is fundamentally hard just different work, for a different
day.

Eric

Subject: [PATCH 00/15] sysfs support for namespaces
Posted by ebiederm on Fri, 04 Jul 2008 00:48:40 GMT

View Forum Message <> Reply to Message

When multiple namespaces are in use we can get multiple kernel objects
with the same name which is currently impossible to represent in sysfs.
In particular directories like /sys/class/net and /sys/kernel/uids have
significant problems.

Not wanting to change the user space interface and wanting to have a simple
implementation where all objects are in the kobject and sysfs trees. The
decision has been made to tag objects with the namespace they live in,
and in a particular mount of sysfs only display objects with the tag
that corresponds to the namespaces in effect when sysfs was mounted.

After the last round of reviews the mount/umount logic is
significantly cleaned up and easier to maintain. From a 10,000
foot view the code and the way it functions has remained the
same since we settled on tagged directories a year or so ago.   I
intend any future cleanups to be as incremental patches on top
of this existing set.

Lack of these patches are keeping the generally complete network
namespace work in 2.6.26 from being used and tested more heavily.
Can we please get the patches merged?

These patches are based off of 2.6.26-rc8 + the -gregkh tree from
last night. Hopefully that means they apply -mm -gregkh and
-linux-next.

Eric

Subject: [PATCH 01/15] kobject: Cleanup kobject_rename and !CONFIG_SYSFS
Posted by ebiederm on Fri, 04 Jul 2008 01:05:28 GMT

View Forum Message <> Reply to Message

It finally dawned on me what the clean fix to sysfs_rename_dir
calling kobject_set_name is. Move the work into kobject_rename
where it belongs. The callers serialize us anyway so this is
safe.

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

```
---
 fs/sysfs/dir.c        |    6 +-----
 include/linux/sysfs.h |    4 +---
 lib/kobject.c         |   17 +++++++++++++++--
 3 files changed, 17 insertions(+), 10 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 8c0e4b9..146b86a 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -799,16 +799,12 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  if (!new_dentry)
   goto out_unlock;

- /* rename kobject and sysfs_dirent */
+ /* rename sysfs_dirent */
  error = -ENOMEM;
  new_name = dup_name = kstrdup(new_name, GFP_KERNEL);
  if (!new_name)
   goto out_unlock;

- error = kobject_set_name(kobj, "%s", new_name);
- if (error)
-  goto out_unlock;
-
```

```
  dup_name = sd->s_name;
  sd->s_name = new_name;

diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
index 84d92bb..f7e43ed 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -20,8 +20,6 @@
 struct kobject;
 struct module;

-extern int kobject_set_name(struct kobject *kobj, const char *name, ...)
-        __attribute__((format(printf, 2, 3)));
 /* FIXME
  * The *owner field is no longer used, but leave around
  * until the tree gets cleaned up fully.
@@ -140,7 +138,7 @@ static inline void sysfs_remove_dir(struct kobject *kobj)

 static inline int sysfs_rename_dir(struct kobject *kobj, const char *new_name)
 {
- return kobject_set_name(kobj, "%s", new_name);
+ return 0;
 }

 static inline int sysfs_move_dir(struct kobject *kobj,
diff --git a/lib/kobject.c b/lib/kobject.c
index 829b839..49b3bc4 100644
--- a/lib/kobject.c
+++ b/lib/kobject.c
@@ -451,6 +451,7 @@ int kobject_rename(struct kobject *kobj, const char *new_name)
 {
  int error = 0;
  const char *devpath = NULL;
+ const char *dup_name = NULL, *name;
  char *devpath_string = NULL;
  char *envp[2];

@@ -474,15 +475,27 @@ int kobject_rename(struct kobject *kobj, const char *new_name)
  envp[0] = devpath_string;
  envp[1] = NULL;

+ name = dup_name = kstrdup(new_name, GFP_KERNEL);
+ if (!name) {
+  error = -ENOMEM;
+  goto out;
+ }
+
  error = sysfs_rename_dir(kobj, new_name);
```

```
+ if (error)
+  goto out;
+
+ /* Install the new kobject name */
+ dup_name = kobj->name;
+ kobj->name = name;

  /* This function is mostly/only used for network interface.
   * Some hotplug package track interfaces by their name and
   * therefore want to know when the name is changed by the user. */
- if (!error)
-  kobject_uevent_env(kobj, KOBJ_MOVE, envp);
+ kobject_uevent_env(kobj, KOBJ_MOVE, envp);

 out:
+ kfree(dup_name);
  kfree(devpath_string);
  kfree(devpath);
  kobject_put(kobj);
--
1.5.3.rc6.17.g1911
```

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

Subject: Re: [PATCH 00/15] sysfs support for namespaces
Posted by ebiederm on Fri, 04 Jul 2008 01:27:29 GMT

View Forum Message <> Reply to Message

I should mention patches 1-12 are the core of the work.

Patches 13-15 are the users. Included in complete form primarily to
aid review.

Eric

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

Subject: Re: [PATCH 01/15] kobject: Cleanup kobject_rename and !CONFIG_SYSFS
Posted by Tejun Heo on Fri, 04 Jul 2008 06:33:23 GMT

View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> It finally dawned on me what the clean fix to sysfs_rename_dir
> calling kobject_set_name is. Move the work into kobject_rename
> where it belongs. The callers serialize us anyway so this is
> safe.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

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

--
tejun

Subject: Re: [PATCH 00/15] sysfs support for namespaces
Posted by ebiederm on Sun, 06 Jul 2008 04:42:57 GMT

View Forum Message <> Reply to Message

> These patches are based off of 2.6.26-rc8 + the -gregkh tree from
> last night. Hopefully that means they apply -mm -gregkh and
> -linux-next.

A quick update.   My patchset conflicts with the recently added
  driver-core-suppress-sysfs-warnings-for-device_rename.patch

> driver core: Suppress sysfs warnings for device_rename().
>
> Renaming network devices to an already existing name is not
> something we want sysfs to print a scary warning for, since the
> callers can deal with this correctly. So let's introduce
> sysfs_create_link_nowarn() which gets rid of the common warning.

This patch is unnecessary as that path is never exercised anymore.
as: dev_change_name returns early in the case of a noop rename.

In addition my introduction sysfs_rename_link handles this case
cleanly by first removing the old link and then creating the new
link. Preventing false positives when the link names are the same.

So it should be safe to drop Cornelia patch without a reoccurance
of scary errors.

Eric

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

Subject: Re: [PATCH 00/15] sysfs support for namespaces
Posted by Cornelia Huck on Mon, 07 Jul 2008 11:41:01 GMT
View Forum Message <> Reply to Message

On Sat, 05 Jul 2008 21:42:57 -0700,
ebiederm@xmission.com (Eric W. Biederman) wrote:

>
> > These patches are based off of 2.6.26-rc8 + the -gregkh tree from
> > last night. Hopefully that means they apply -mm -gregkh and
> > -linux-next.
>
> A quick update.   My patchset conflicts with the recently added
>   driver-core-suppress-sysfs-warnings-for-device_rename.patch
>
> > driver core: Suppress sysfs warnings for device_rename().
> >
> > Renaming network devices to an already existing name is not
> > something we want sysfs to print a scary warning for, since the
> > callers can deal with this correctly. So let's introduce
> > sysfs_create_link_nowarn() which gets rid of the common warning.
>
> This patch is unnecessary as that path is never exercised anymore.
> as: dev_change_name returns early in the case of a noop rename.

My impression was that the networking folks didn't want any warnings for
renaming failures, not just not for renaming a device to the same name.

>
> In addition my introduction sysfs_rename_link handles this case
> cleanly by first removing the old link and then creating the new
> link. Preventing false positives when the link names are the same.

sysfs_rename_link() looks cleaner, I agree.

>
> So it should be safe to drop Cornelia patch without a reoccurance
> of scary errors.

Hm, the description looks badly worded - I unfortunately left the old
text unchanged when I respun the patch :(The patch re-introduces the
warning in sysfs_add_one() which had been removed in the meanwhile and
makes device_rename() use a non-warning version. I still think we want
a warning for the general case since this is usually caused be some
problems in the calling code (and the alternative would be to add
checks to all callers.)

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

Subject: Re: [PATCH 00/15] sysfs support for namespaces
Posted by ebiederm on Mon, 07 Jul 2008 12:22:43 GMT

View Forum Message <> Reply to Message

Cornelia Huck <cornelia.huck@de.ibm.com> writes:

> My impression was that the networking folks didn't want any warnings for
> renaming failures, not just not for renaming a device to the same name.

Which would be reasonable. Because all of the checks have been done
before sysfs is called so if sysfs sees a problem it is a sysfs bug.

>> In addition my introduction sysfs_rename_link handles this case
>> cleanly by first removing the old link and then creating the new
>> link. Preventing false positives when the link names are the same.
>
> sysfs_rename_link() looks cleaner, I agree.
>
>>
>> So it should be safe to drop Cornelia patch without a reoccurance
>> of scary errors.
>
> Hm, the description looks badly worded - I unfortunately left the old
> text unchanged when I respun the patch :(The patch re-introduces the
> warning in sysfs_add_one() which had been removed in the meanwhile and
> makes device_rename() use a non-warning version. I still think we want
> a warning for the general case since this is usually caused be some
> problems in the calling code (and the alternative would be to add
> checks to all callers.)

Right. We just need to get the sysfs paths clean enough that we don't
emit false positives. I think I have accomplished that for rename.

Eric
____