Subject: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by ebiederm on Tue, 07 Aug 2007 21:06:21 GMT
View Forum Message <> Reply to Message

The following patchset applies on top of the last round of
sysfs cleanups that Tejun sent out on the 2nd.

My target with this patchset is to support sysfs directories
with a tag on struct sysfs_dirent making them visible only
on selected mounts of sysfs.

After going around and around the different possibilities I
believe I have finally found something that works and is
reasonably maintainable. I believe I have achieved that
with only introducing some extra complexity in a few very localized
places.

The worst part is the code to support multiple superblocks and thus
multiple dentry tress for sysfs. I had allocate a linked list in
sysfs_move_dir for all of the possible dentries I would need to call
d_move on. Bleh. It works, it is correct and it is an atomic
rename.

Eric

sysfs: Move all of inode initialization into sysfs_init_inode
sysfs: Remove sysfs_instantiate
sysfs: Use kill_anon_super
sysfs: Make sysfs_mount static
sysfs: In sysfs_lookup don't open code sysfs_find_dirent
sysfs: Simplify readdir.
sysfs: Rewrite sysfs_drop_dentry.
sysfs: Implement __sysfs_get_dentry
sysfs: Move sysfs_get_dentry below __sysfs_get_dentry
sysfs: Rewrite sysfs_get_dentry in terms of __sysfs_get_dentry
sysfs: Remove s_dentry
sysfs: Introduce sysfs_rename_mutex
sysfs: Simply sysfs_get_dentry
sysfs: Don't use lookup_one_len_kern
vfs: Remove lookup_one_len_kern
sysfs: Support for preventing unmounts.
sysfs: Rewrite rename in terms of sysfs dirents
sysfs: Rewrite sysfs_move_dir in terms of sysfs dirents
sysfs: sysfs_get_dentry add a sb parameter
sysfs: Rename Support multiple superblocks
sysfs: sysfs_chmod_file handle multiple superblocks
sysfs: sysfs_uptdate_file handle multiple superblocks

sysfs: Implement sysfs tagged directory support.
sysfs: Implement sysfs_delete_link and sysfs_rename_link
driver core: Implement tagged directory support for device classes.

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

---

Subject: [PATCH 01/25] sysfs: Move all of inode initialization into sysfs_init_inode
Posted by ebiederm on Tue, 07 Aug 2007 21:08:07 GMT
View Forum Message <> Reply to Message

Signed-off-by: "Eric W. Biederman" <ebiederm@xmission.com>
---
 fs/sysfs/dir.c | 37 ----------------------------------
 fs/sysfs/inode.c | 48 ++---
 fs/sysfs/mount.c | 5 -----
 3 files changed, 45 insertions(+), 45 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 3e5acc4..ae06f4a 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -746,24 +746,12 @@ int sysfs_create_dir(struct kobject * kobj)
  return error;
 }

-static int sysfs_count_nlink(struct sysfs_dirent *sd)
-{
- struct sysfs_dirent *child;
- int nr = 0;
-
- for (child = sd->s_children; child; child = child->s_sibling)
- if (sysfs_type(child) == SYSFS_DIR)
-   nr++;
- return nr + 2;
-}
-
 static struct dentry * sysfs_lookup(struct inode *dir, struct dentry *dentry,
     struct nameidata *nd)
 {
  struct dentry *ret = NULL;
  struct sysfs_dirent *parent_sd = dentry->d_parent->d_fsdata;
  struct sysfs_dirent *sd;
- struct bin_attribute *bin_attr;
  struct inode *inode;

```
  mutex_lock(&sysfs_mutex);
@@ -783,31 +771,6 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry
*dentry,
   goto out_unlock;
  }

- if (inode->i_state & I_NEW) {
-  /* initialize inode according to type */
-  switch (sysfs_type(sd)) {
-  case SYSFS_DIR:
-   inode->i_op = &sysfs_dir_inode_operations;
-   inode->i_fop = &sysfs_dir_operations;
-   inode->i_nlink = sysfs_count_nlink(sd);
-   break;
-  case SYSFS_KOBJ_ATTR:
-   inode->i_size = PAGE_SIZE;
-   inode->i_fop = &sysfs_file_operations;
-   break;
-  case SYSFS_KOBJ_BIN_ATTR:
-   bin_attr = sd->s_elem.bin_attr.bin_attr;
-   inode->i_size = bin_attr->size;
-   inode->i_fop = &bin_fops;
-   break;
-  case SYSFS_KOBJ_LINK:
-   inode->i_op = &sysfs_symlink_inode_operations;
-   break;
-  default:
-   BUG();
-  }
- }
-
  sysfs_instantiate(dentry, inode);
  sysfs_attach_dentry(sd, dentry);

diff --git a/fs/sysfs/inode.c b/fs/sysfs/inode.c
index e74224e..aefc346 100644
--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -123,8 +123,22 @@ static inline void set_inode_attr(struct inode * inode, struct iattr * iattr)
  */
 static struct lock_class_key sysfs_inode_imutex_key;

+static int sysfs_count_nlink(struct sysfs_dirent *sd)
+{
+ struct sysfs_dirent *child;
+ int nr = 0;
+
+ for (child = sd->s_children; child; child = child->s_sibling)
```

```
+  if (sysfs_type(child) == SYSFS_DIR)
+   nr++;
+
+ return nr + 2;
+}
+
 static void sysfs_init_inode(struct sysfs_dirent *sd, struct inode *inode)
 {
+ struct bin_attribute *bin_attr;
+
  inode->i_blocks = 0;
  inode->i_mapping->a_ops = &sysfs_aops;
  inode->i_mapping->backing_dev_info = &sysfs_backing_dev_info;
@@ -140,6 +154,37 @@ static void sysfs_init_inode(struct sysfs_dirent *sd, struct inode *inode)
   set_inode_attr(inode, sd->s_iattr);
  } else
   set_default_inode_attr(inode, sd->s_mode);
+
+
+ /* initialize inode according to type */
+ switch (sysfs_type(sd)) {
+ case SYSFS_ROOT:
+  inode->i_op = &sysfs_dir_inode_operations;
+  inode->i_fop = &sysfs_dir_operations;
+  inc_nlink(inode); /* directory, account for "." */
+  break;
+ case SYSFS_DIR:
+  inode->i_op = &sysfs_dir_inode_operations;
+  inode->i_fop = &sysfs_dir_operations;
+  inode->i_nlink = sysfs_count_nlink(sd);
+  break;
+ case SYSFS_KOBJ_ATTR:
+  inode->i_size = PAGE_SIZE;
+  inode->i_fop = &sysfs_file_operations;
+  break;
+ case SYSFS_KOBJ_BIN_ATTR:
+  bin_attr = sd->s_elem.bin_attr.bin_attr;
+  inode->i_size = bin_attr->size;
+  inode->i_fop = &bin_fops;
+  break;
+ case SYSFS_KOBJ_LINK:
+  inode->i_op = &sysfs_symlink_inode_operations;
+  break;
+ default:
+  BUG();
+ }
+
+ unlock_new_inode(inode);
```

```
 }

 /**
@@ -181,9 +226,6 @@ void sysfs_instantiate(struct dentry *dentry, struct inode *inode)
 {
  BUG_ON(!dentry || dentry->d_inode);

- if (inode->i_state & I_NEW)
-  unlock_new_inode(inode);
-
  d_instantiate(dentry, inode);
 }

diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index 0c016e1..919eaaf 100644
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -50,11 +50,6 @@ static int sysfs_fill_super(struct super_block *sb, void *data, int silent)
   return -ENOMEM;
  }

- inode->i_op = &sysfs_dir_inode_operations;
- inode->i_fop = &sysfs_dir_operations;
- inc_nlink(inode); /* directory, account for "." */
- unlock_new_inode(inode);
-
  /* instantiate and link root dentry */
  root = d_alloc_root(inode);
  if (!root) {
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 02/25] sysfs: Remove sysfs_instantiate
Posted by ebiederm on Tue, 07 Aug 2007 21:08:50 GMT
View Forum Message <> Reply to Message

Now that sysfs_get_inode is dropping the inode lock
we no longer have a need from sysfs_instantiate.

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

```
 fs/sysfs/dir.c   |    2 +-
```

```
 fs/sysfs/inode.c |   17 -----------------
 fs/sysfs/sysfs.h |    1 -
 3 files changed, 1 insertions(+), 19 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index ae06f4a..55d8299 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -771,7 +771,7 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry *dentry,
   goto out_unlock;
  }

- sysfs_instantiate(dentry, inode);
+ d_instantiate(dentry, inode);
  sysfs_attach_dentry(sd, dentry);

  out_unlock:
diff --git a/fs/sysfs/inode.c b/fs/sysfs/inode.c
index aefc346..70a2420 100644
--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -212,23 +212,6 @@ struct inode * sysfs_get_inode(struct sysfs_dirent *sd)
  return inode;
 }

-/**
- * sysfs_instantiate - instantiate dentry
- * @dentry: dentry to be instantiated
- * @inode: inode associated with @sd
- *
- * Unlock @inode if locked and instantiate @dentry with @inode.
- *
- * LOCKING:
- * None.
- */
-void sysfs_instantiate(struct dentry *dentry, struct inode *inode)
-{
- BUG_ON(!dentry || dentry->d_inode);
-
- d_instantiate(dentry, inode);
-}
-
 int sysfs_hash_and_remove(struct sysfs_dirent *dir_sd, const char *name)
 {
  struct sysfs_addrm_cxt acxt;
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index 0436754..8a0aea1 100644
--- a/fs/sysfs/sysfs.h
```

```
+++ b/fs/sysfs/sysfs.h
@@ -69,7 +69,6 @@ extern void sysfs_remove_one(struct sysfs_addrm_cxt *acxt,
 extern void sysfs_addrm_finish(struct sysfs_addrm_cxt *acxt);

 extern struct inode * sysfs_get_inode(struct sysfs_dirent *sd);
-extern void sysfs_instantiate(struct dentry *dentry, struct inode *inode);

 extern void release_sysfs_dirent(struct sysfs_dirent * sd);
 extern struct sysfs_dirent *sysfs_find_dirent(struct sysfs_dirent *parent_sd,
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 03/25] sysfs: Use kill_anon_super

Posted by ebiederm on Tue, 07 Aug 2007 21:10:27 GMT

View Forum Message <> Reply to Message

Since sysfs no longer stores fs directory information in the dcache
on a permanent basis kill_litter_super it is inappropriate and actively
wrong. It will decrement the count on all dentries left in the
dcache before trying to free them.

At the moment this is not biting us only because we never unmount sysfs.

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

```
---
 fs/sysfs/mount.c |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)

diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index 919eaaf..d1094c6 100644
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -72,7 +72,7 @@ static int sysfs_get_sb(struct file_system_type *fs_type,
 static struct file_system_type sysfs_fs_type = {
   .name  = "sysfs",
   .get_sb  = sysfs_get_sb,
- .kill_sb = kill_litter_super,
+ .kill_sb = kill_anon_super,
 };

 int __init sysfs_init(void)
--
```

1.5.1.1.181.g2de0

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

---

Subject: [PATCH 04/25] sysfs: Make sysfs_mount static
Posted by ebiederm on Tue, 07 Aug 2007 21:11:16 GMT
View Forum Message <> Reply to Message

This patch modifies the users of sysfs_mount to use sysfs_root
instead (which is what they are looking for). It then
makes sysfs_mount static to keep people from using it
by accident.

The net result is slightly faster and cleaner code.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
fs/sysfs/dir.c | 4 +---
fs/sysfs/mount.c | 2 +-
fs/sysfs/symlink.c | 7 ++++----
fs/sysfs/sysfs.h | 1 -
4 files changed, 5 insertions(+), 9 deletions(-)

```
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 55d8299..39df3ce 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -735,10 +735,8 @@ int sysfs_create_dir(struct kobject * kobj)

  if (kobj->parent)
   parent_sd = kobj->parent->sd;
- else if (sysfs_mount && sysfs_mount->mnt_sb)
-  parent_sd = sysfs_mount->mnt_sb->s_root->d_fsdata;
  else
-  return -EFAULT;
+  parent_sd = &sysfs_root;

  error = create_dir(kobj, parent_sd, kobject_name(kobj), &sd);
  if (!error)
diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index d1094c6..9fae7d5 100644
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -15,7 +15,7 @@
```

```
 /* Random magic number */
 #define SYSFS_MAGIC 0x62656572

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

diff --git a/fs/sysfs/symlink.c b/fs/sysfs/symlink.c
index f77ad61..46f8fd4 100644
--- a/fs/sysfs/symlink.c
+++ b/fs/sysfs/symlink.c
@@ -60,10 +60,9 @@ int sysfs_create_link(struct kobject * kobj, struct kobject * target, const
char

  BUG_ON(!name);

- if (!kobj) {
-  if (sysfs_mount && sysfs_mount->mnt_sb)
-   parent_sd = sysfs_mount->mnt_sb->s_root->d_fsdata;
- } else
+ if (!kobj)
+  parent_sd = &sysfs_root;
+ else
   parent_sd = kobj->sd;

  error = -EFAULT;
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index 8a0aea1..77253aa 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -51,7 +51,6 @@ struct sysfs_addrm_cxt {
  int   cnt;
 };

-extern struct vfsmount * sysfs_mount;
 extern struct sysfs_dirent sysfs_root;
 extern struct kmem_cache *sysfs_dir_cachep;

--
1.5.1.1.181.g2de0
```

Subject: [PATCH 05/25] sysfs: In sysfs_lookup don't open code sysfs_find_dirent
Posted by ebiederm on Tue, 07 Aug 2007 21:12:02 GMT
View Forum Message <> Reply to Message

This is a small cleanup patch that makes the code just
a little bit cleaner.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/dir.c | 4 +---
 1 files changed, 1 insertions(+), 3 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 39df3ce..2721e36 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -754,9 +754,7 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry *dentry,

  mutex_lock(&sysfs_mutex);

- for (sd = parent_sd->s_children; sd; sd = sd->s_sibling)
- if (sysfs_type(sd) && !strcmp(sd->s_name, dentry->d_name.name))
-   break;
+ sd = sysfs_find_dirent(parent_sd, dentry->d_name.name);

  /* no such entry */
  if (!sd)
--
1.5.1.1.181.g2de0

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

Subject: [PATCH 06/25] sysfs: Simplify readdir.
Posted by ebiederm on Tue, 07 Aug 2007 21:13:14 GMT
View Forum Message <> Reply to Message

At some point someone wrote sysfs_readdir to insert a cursor
into the list of sysfs_dirents to ensure that sysfs_readdir would
restart properly. That works but it is complex code and tends
to be expensive.

The same effect can be achieved by keeping the sysfs_dirents in
inode order and using the inode number as the f_pos. Then
when we restart we just have to find the first dirent whose inode

number is equal or greater then the last sysfs_dirent we attempted
to return.

Removing the sysfs directory cursor also allows the remove of
all of the mysterious checks for sysfs_type(sd) != 0.   Which
were nonbovious checks to see if a cursor was in a directory list.

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

```
---
 fs/sysfs/dir.c |  175 ++++++++++++++-----------------------------------------
 1 files changed, 44 insertions(+), 131 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 2721e36..ef99883 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -33,10 +33,20 @@ static DEFINE_IDA(sysfs_ino_ida);
 static void sysfs_link_sibling(struct sysfs_dirent *sd)
 {
  struct sysfs_dirent *parent_sd = sd->s_parent;
+ struct sysfs_dirent **pos;

  BUG_ON(sd->s_sibling);
- sd->s_sibling = parent_sd->s_children;
- parent_sd->s_children = sd;
+
+ /* Store directory entries in order by ino.  This allows
+  * readdir to properly restart without having to add a
+  * cursor into the s_children list.
+  */
+ for (pos = &parent_sd->s_children; *pos; pos = &(*pos)->s_sibling) {
+  if (sd->s_ino < (*pos)->s_ino)
+   break;
+ }
+ sd->s_sibling = *pos;
+ *pos = sd;
 }

 /**
@@ -657,7 +667,7 @@ struct sysfs_dirent *sysfs_find_dirent(struct sysfs_dirent *parent_sd,
  struct sysfs_dirent *sd;

  for (sd = parent_sd->s_children; sd; sd = sd->s_sibling)
-  if (sysfs_type(sd) && !strcmp(sd->s_name, name))
+  if (!strcmp(sd->s_name, name))
    return sd;
  return NULL;
 }
```

```
@@ -809,7 +819,7 @@ static void __sysfs_remove_dir(struct sysfs_dirent *dir_sd)
  while (*pos) {
   struct sysfs_dirent *sd = *pos;

-  if (sysfs_type(sd) && sysfs_type(sd) != SYSFS_DIR)
+  if (sysfs_type(sd) != SYSFS_DIR)
    sysfs_remove_one(&acxt, sd);
   else
    pos = &(*pos)->s_sibling;
@@ -974,37 +984,6 @@ again:
  return error;
 }

-static int sysfs_dir_open(struct inode *inode, struct file *file)
-{
- struct dentry * dentry = file->f_path.dentry;
- struct sysfs_dirent * parent_sd = dentry->d_fsdata;
- struct sysfs_dirent * sd;
-
- sd = sysfs_new_dirent("_DIR_", 0, 0);
- if (sd) {
-  mutex_lock(&sysfs_mutex);
-  sd->s_parent = sysfs_get(parent_sd);
-  sysfs_link_sibling(sd);
-  mutex_unlock(&sysfs_mutex);
- }
-
- file->private_data = sd;
- return sd ? 0 : -ENOMEM;
-}
-
-static int sysfs_dir_close(struct inode *inode, struct file *file)
-{
- struct sysfs_dirent * cursor = file->private_data;
-
- mutex_lock(&sysfs_mutex);
- sysfs_unlink_sibling(cursor);
- mutex_unlock(&sysfs_mutex);
-
- release_sysfs_dirent(cursor);
-
- return 0;
-}
-
 /* Relationship between s_mode and the DT_xxx types */
 static inline unsigned char dt_type(struct sysfs_dirent *sd)
 {
@@ -1015,117 +994,51 @@ 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 sysfs_dirent *cursor = filp->private_data;
- struct sysfs_dirent **pos;
+ struct sysfs_dirent *pos;
  ino_t ino;
- int i = filp->f_pos;

- switch (i) {
-  case 0:
-   ino = parent_sd->s_ino;
-   if (filldir(dirent, ".", 1, i, ino, DT_DIR) < 0)
-    break;
+ if (filp->f_pos == 0) {
+  ino = parent_sd->s_ino;
+  if (filldir(dirent, ".", 1, filp->f_pos, ino, DT_DIR) == 0)
    filp->f_pos++;
-   i++;
-   /* fallthrough */
-  case 1:
-   if (parent_sd->s_parent)
-    ino = parent_sd->s_parent->s_ino;
-   else
-    ino = parent_sd->s_ino;
-   if (filldir(dirent, "..", 2, i, ino, DT_DIR) < 0)
-    break;
+ }
+ if (filp->f_pos == 1) {
+  if (parent_sd->s_parent)
+   ino = parent_sd->s_parent->s_ino;
+  else
+   ino = parent_sd->s_ino;
+  if (filldir(dirent, "..", 2, filp->f_pos, ino, DT_DIR) == 0)
    filp->f_pos++;
-   i++;
-   /* fallthrough */
-  default:
-   mutex_lock(&sysfs_mutex);
-
-   pos = &parent_sd->s_children;
-   while (*pos != cursor)
-    pos = &(*pos)->s_sibling;
-
-   /* unlink cursor */
-   *pos = cursor->s_sibling;
-
-   if (filp->f_pos == 2)
```

```
-    pos = &parent_sd->s_children;
-
-   for ( ; *pos; pos = &(*pos)->s_sibling) {
-    struct sysfs_dirent *next = *pos;
-    const char * name;
-    int len;
-
-    if (!sysfs_type(next))
-     continue;
-
-    name = next->s_name;
-    len = strlen(name);
-    ino = next->s_ino;
-
-    if (filldir(dirent, name, len, filp->f_pos, ino,
-       dt_type(next)) < 0)
-     break;
-
-    filp->f_pos++;
-   }
+ }
+ if ((filp->f_pos > 1) && (filp->f_pos < UINT_MAX)) {
+  mutex_lock(&sysfs_mutex);

-   /* put cursor back in */
-   cursor->s_sibling = *pos;
-   *pos = cursor;
+  /* Skip the dentries we have already reported */
+  pos = parent_sd->s_children;
+  while (pos && (filp->f_pos > pos->s_ino))
+   pos = pos->s_sibling;

-   mutex_unlock(&sysfs_mutex);
- }
- return 0;
-}
+  for ( ; pos; pos = pos->s_sibling) {
+   const char * name;
+   int len;

-static loff_t sysfs_dir_lseek(struct file * file, loff_t offset, int origin)
-{
- struct dentry * dentry = file->f_path.dentry;
+   name = pos->s_name;
+   len = strlen(name);
+   filp->f_pos = ino = pos->s_ino;

- switch (origin) {
```

```
-  case 1:
-   offset += file->f_pos;
-  case 0:
-   if (offset >= 0)
+   if (filldir(dirent, name, len, filp->f_pos, ino,
+       dt_type(pos)) < 0)
     break;
-  default:
-   return -EINVAL;
- }
- if (offset != file->f_pos) {
-  mutex_lock(&sysfs_mutex);
-
-  file->f_pos = offset;
-  if (file->f_pos >= 2) {
-   struct sysfs_dirent *sd = dentry->d_fsdata;
-   struct sysfs_dirent *cursor = file->private_data;
-   struct sysfs_dirent **pos;
-   loff_t n = file->f_pos - 2;
-
-   sysfs_unlink_sibling(cursor);
-
-   pos = &sd->s_children;
-   while (n && *pos) {
-    struct sysfs_dirent *next = *pos;
-    if (sysfs_type(next))
-     n--;
-    pos = &(*pos)->s_sibling;
-   }
-
-   cursor->s_sibling = *pos;
-   *pos = cursor;
   }
-
+  if (!pos)
+   filp->f_pos = UINT_MAX;
   mutex_unlock(&sysfs_mutex);
  }
-
- return offset;
+ return 0;
 }

+
 const struct file_operations sysfs_dir_operations = {
- .open  = sysfs_dir_open,
- .release = sysfs_dir_close,
- .llseek  = sysfs_dir_lseek,
```

```
  .read  = generic_read_dir,
  .readdir = sysfs_readdir,
 };
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 07/25] sysfs: Rewrite sysfs_drop_dentry.
Posted by ebiederm on Tue, 07 Aug 2007 21:14:56 GMT
View Forum Message <> Reply to Message

Currently we find the dentry to drop by looking at sd->s_dentry.
We can just as easily accomplish the same task by looking up the
sysfs inode and finding all of the dentries from there, with the
added bonus that we don't need to play with the sysfs_assoc_lock.

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

```
---
 fs/sysfs/dir.c |   53 ++++++++++++++++++++++++++++++++----------------------
 1 files changed, 26 insertions(+), 27 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index ef99883..6ca4382 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -563,50 +563,49 @@ void sysfs_remove_one(struct sysfs_addrm_cxt *acxt, struct
sysfs_dirent *sd)
  * Drop dentry for @sd.  @sd must have been unlinked from its
  * parent on entry to this function such that it can't be looked
  * up anymore.
- *
- * @sd->s_dentry which is protected with sysfs_assoc_lock points
- * to the currently associated dentry but we're not holding a
- * reference to it and racing with dput().  Grab dcache_lock and
- * verify dentry before dropping it.  If @sd->s_dentry is NULL or
- * dput() beats us, no need to bother.
  */
 static void sysfs_drop_dentry(struct sysfs_dirent *sd)
 {
- struct dentry *dentry = NULL;
  struct inode *inode;
+ struct dentry *dentry;
+
```

```
+ inode = ilookup(sysfs_sb, sd->s_ino);
+ if (!inode)
+  return;

- /* We're not holding a reference to ->s_dentry dentry but the
-  * field will stay valid as long as sysfs_assoc_lock is held.
+ /* Drop any existing dentries associated with sd.
+  *
+  * For the dentry to be properly freed we need to grab a
+  * reference to the dentry under the dcache lock,  unhash it,
+  * and then put it.  The playing with the dentry count allows
+  * dput to immediately free the dentry  if it is not in use.
   */
- spin_lock(&sysfs_assoc_lock);
+repeat:
  spin_lock(&dcache_lock);
-
- /* drop dentry if it's there and dput() didn't kill it yet */
- if (sd->s_dentry && sd->s_dentry->d_inode) {
-  dentry = dget_locked(sd->s_dentry);
+ list_for_each_entry(dentry, &inode->i_dentry, d_alias) {
+  if (d_unhashed(dentry))
+   continue;
+  dget_locked(dentry);
   spin_lock(&dentry->d_lock);
   __d_drop(dentry);
   spin_unlock(&dentry->d_lock);
+  spin_unlock(&dcache_lock);
+  dput(dentry);
+  goto repeat;
  }
-
  spin_unlock(&dcache_lock);
- spin_unlock(&sysfs_assoc_lock);
-
- dput(dentry);

  /* adjust nlink and update timestamp */
- inode = ilookup(sysfs_sb, sd->s_ino);
- if (inode) {
-  mutex_lock(&inode->i_mutex);
+ mutex_lock(&inode->i_mutex);

-  inode->i_ctime = CURRENT_TIME;
+ inode->i_ctime = CURRENT_TIME;
+ drop_nlink(inode);
+ if (sysfs_type(sd) == SYSFS_DIR)
   drop_nlink(inode);
```

```
-  if (sysfs_type(sd) == SYSFS_DIR)
-   drop_nlink(inode);

-  mutex_unlock(&inode->i_mutex);
-  iput(inode);
- }
+ mutex_unlock(&inode->i_mutex);
+
+ iput(inode);
 }

 /**
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 08/25] sysfs: Implement __sysfs_get_dentry
Posted by ebiederm on Tue, 07 Aug 2007 21:16:19 GMT
View Forum Message <> Reply to Message

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

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

```
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 6ca4382..2caa5e0 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -848,6 +848,44 @@ void sysfs_remove_dir(struct kobject * kobj)
  __sysfs_remove_dir(sd);
 }

+/**
+ * __sysfs_get_dentry - get dentry for the given sysfs_dirent
+ * @sb: superblock of the dentry to return
+ * @sd: sysfs_dirent of interest
+ *
+ * Get dentry for @sd.  Only return a dentry if one currently
+ * exists.
```

```
+ *
+ * LOCKING:
+ * Kernel thread context (may sleep)
+ *
+ * RETURNS:
+ * Pointer to found dentry on success, NULL on failure.
+ */
+static struct dentry *__sysfs_get_dentry(struct super_block *sb, struct sysfs_dirent *sd)
+{
+ struct inode *inode;
+ struct dentry *dentry = NULL;
+
+ inode = ilookup5_nowait(sysfs_sb, sd->s_ino, sysfs_ilookup_test, sd);
+ if (inode && !(inode->i_state & I_NEW)) {
+  struct dentry *alias;
+  spin_lock(&dcache_lock);
+  list_for_each_entry(alias, &inode->i_dentry, d_alias) {
+   if (!IS_ROOT(alias) && d_unhashed(alias))
+    continue;
+   if (alias->d_sb != sb)
+    continue;
+   dentry = alias;
+   dget_locked(dentry);
+   break;
+  }
+  spin_unlock(&dcache_lock);
+ }
+ iput(inode);
+ return dentry;
+}
+
 int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
 {
  struct sysfs_dirent *sd;
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 09/25] sysfs: Move sysfs_get_dentry below __sysfs_get_dentry
Posted by ebiederm on Tue, 07 Aug 2007 21:17:33 GMT
View Forum Message <> Reply to Message

sysfs_get_dentry is higher in fs/sysfs/dir.c then is needed and it the

dependencies get simpler if we move it down in the file to where I
have placed __sysfs_get_dentry. So this patch just moves
sysfs_get_dentry so code movement doesn't get confused with later code
changes.

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

```
---
 fs/sysfs/dir.c |  196 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++------------------------------------------------------------------------------------------------
 1 files changed, 98 insertions(+), 98 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 2caa5e0..59a9ce8 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -73,104 +73,6 @@ static void sysfs_unlink_sibling(struct sysfs_dirent *sd)
 }

 /**
- * sysfs_get_dentry - get dentry for the given sysfs_dirent
- * @sd: sysfs_dirent of interest
- *
- * Get dentry for @sd.  Dentry is looked up if currently not
- * present.  This function climbs sysfs_dirent tree till it
- * reaches a sysfs_dirent with valid dentry attached and descends
- * down from there looking up dentry for each step.
- *
- * LOCKING:
- * Kernel thread context (may sleep)
- *
- * RETURNS:
- * Pointer to found dentry on success, ERR_PTR() value on error.
- */
-struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd)
-{
- struct sysfs_dirent *cur;
- struct dentry *parent_dentry, *dentry;
- int i, depth;
-
- /* Find the first parent which has valid s_dentry and get the
-  * dentry.
-  */
- mutex_lock(&sysfs_mutex);
- restart0:
- spin_lock(&sysfs_assoc_lock);
- restart1:
- spin_lock(&dcache_lock);
-
- dentry = NULL;
```

```
- depth = 0;
- cur = sd;
- while (!cur->s_dentry || !cur->s_dentry->d_inode) {
-  if (cur->s_flags & SYSFS_FLAG_REMOVED) {
-   dentry = ERR_PTR(-ENOENT);
-   depth = 0;
-   break;
-  }
-  cur = cur->s_parent;
-  depth++;
- }
- if (!IS_ERR(dentry))
-  dentry = dget_locked(cur->s_dentry);
-
- spin_unlock(&dcache_lock);
- spin_unlock(&sysfs_assoc_lock);
-
- /* from the found dentry, look up depth times */
- while (depth--) {
-  /* find and get depth'th ancestor */
-  for (cur = sd, i = 0; cur && i < depth; i++)
-   cur = cur->s_parent;
-
-  /* This can happen if tree structure was modified due
-   * to move/rename.  Restart.
-   */
-  if (i != depth) {
-   dput(dentry);
-   goto restart0;
-  }
-
-  sysfs_get(cur);
-
-  mutex_unlock(&sysfs_mutex);
-
-  /* look it up */
-  parent_dentry = dentry;
-  dentry = lookup_one_len_kern(cur->s_name, parent_dentry,
-          strlen(cur->s_name));
-  dput(parent_dentry);
-
-  if (IS_ERR(dentry)) {
-   sysfs_put(cur);
-   return dentry;
-  }
-
-  mutex_lock(&sysfs_mutex);
-  spin_lock(&sysfs_assoc_lock);
```

```
-
-  /* This, again, can happen if tree structure has
-   * changed and we looked up the wrong thing.  Restart.
-   */
-  if (cur->s_dentry != dentry) {
-   dput(dentry);
-   sysfs_put(cur);
-   goto restart1;
-  }
-
-  spin_unlock(&sysfs_assoc_lock);
-
-  sysfs_put(cur);
- }
-
- mutex_unlock(&sysfs_mutex);
- return dentry;
-}
-
-/**
  * sysfs_get_active - get an active reference to sysfs_dirent
  * @sd: sysfs_dirent to get an active reference to
  *
@@ -886,6 +788,104 @@ static struct dentry *__sysfs_get_dentry(struct super_block *sb, struct
sysfs_di
  return dentry;
 }

+/**
+ * sysfs_get_dentry - get dentry for the given sysfs_dirent
+ * @sd: sysfs_dirent of interest
+ *
+ * Get dentry for @sd.  Dentry is looked up if currently not
+ * present.  This function climbs sysfs_dirent tree till it
+ * reaches a sysfs_dirent with valid dentry attached and descends
+ * down from there looking up dentry for each step.
+ *
+ * LOCKING:
+ * Kernel thread context (may sleep)
+ *
+ * RETURNS:
+ * Pointer to found dentry on success, ERR_PTR() value on error.
+ */
+struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd)
+{
+ struct sysfs_dirent *cur;
+ struct dentry *parent_dentry, *dentry;
+ int i, depth;
```

```
+
+ /* Find the first parent which has valid s_dentry and get the
+  * dentry.
+  */
+ mutex_lock(&sysfs_mutex);
+ restart0:
+ spin_lock(&sysfs_assoc_lock);
+ restart1:
+ spin_lock(&dcache_lock);
+
+ dentry = NULL;
+ depth = 0;
+ cur = sd;
+ while (!cur->s_dentry || !cur->s_dentry->d_inode) {
+  if (cur->s_flags & SYSFS_FLAG_REMOVED) {
+   dentry = ERR_PTR(-ENOENT);
+   depth = 0;
+   break;
+  }
+  cur = cur->s_parent;
+  depth++;
+ }
+ if (!IS_ERR(dentry))
+  dentry = dget_locked(cur->s_dentry);
+
+ spin_unlock(&dcache_lock);
+ spin_unlock(&sysfs_assoc_lock);
+
+ /* from the found dentry, look up depth times */
+ while (depth--) {
+  /* find and get depth'th ancestor */
+  for (cur = sd, i = 0; cur && i < depth; i++)
+   cur = cur->s_parent;
+
+  /* This can happen if tree structure was modified due
+   * to move/rename.  Restart.
+   */
+  if (i != depth) {
+   dput(dentry);
+   goto restart0;
+  }
+
+  sysfs_get(cur);
+
+  mutex_unlock(&sysfs_mutex);
+
+  /* look it up */
+  parent_dentry = dentry;
```

```
+  dentry = lookup_one_len_kern(cur->s_name, parent_dentry,
+          strlen(cur->s_name));
+  dput(parent_dentry);
+
+  if (IS_ERR(dentry)) {
+   sysfs_put(cur);
+   return dentry;
+  }
+
+  mutex_lock(&sysfs_mutex);
+  spin_lock(&sysfs_assoc_lock);
+
+  /* This, again, can happen if tree structure has
+   * changed and we looked up the wrong thing.  Restart.
+   */
+  if (cur->s_dentry != dentry) {
+   dput(dentry);
+   sysfs_put(cur);
+   goto restart1;
+  }
+
+  spin_unlock(&sysfs_assoc_lock);
+
+  sysfs_put(cur);
+ }
+
+ mutex_unlock(&sysfs_mutex);
+ return dentry;
+}
+
 int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
 {
  struct sysfs_dirent *sd;
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 10/25] sysfs: Rewrite sysfs_get_dentry in terms of __sysfs_get_dentry

Posted by ebiederm on Tue, 07 Aug 2007 21:18:27 GMT

View Forum Message <> Reply to Message

This removes the last major user of s_dentry and makes

```
the locking in sysfs_get_dentry much simpler.  Hopefully
leading to more readable and maintainable code.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/dir.c |   23 ++++++----------------
 1 files changed, 6 insertions(+), 17 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 59a9ce8..adb1b01 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -809,19 +809,15 @@ struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd)
  struct dentry *parent_dentry, *dentry;
  int i, depth;

- /* Find the first parent which has valid s_dentry and get the
-  * dentry.
+ /* Find the first parent which has valid dentry.
   */
  mutex_lock(&sysfs_mutex);
- restart0:
- spin_lock(&sysfs_assoc_lock);
- restart1:
- spin_lock(&dcache_lock);
+ restart:

  dentry = NULL;
  depth = 0;
  cur = sd;
- while (!cur->s_dentry || !cur->s_dentry->d_inode) {
+ while (!(dentry = __sysfs_get_dentry(sysfs_sb, cur))) {
   if (cur->s_flags & SYSFS_FLAG_REMOVED) {
    dentry = ERR_PTR(-ENOENT);
    depth = 0;
@@ -830,11 +826,6 @@ struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd)
   cur = cur->s_parent;
   depth++;
  }
- if (!IS_ERR(dentry))
-  dentry = dget_locked(cur->s_dentry);
-
- spin_unlock(&dcache_lock);
- spin_unlock(&sysfs_assoc_lock);

  /* from the found dentry, look up depth times */
  while (depth--) {
@@ -847,7 +838,7 @@ struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd)
```

```
    */
   if (i != depth) {
    dput(dentry);
-   goto restart0;
+   goto restart;
   }

   sysfs_get(cur);
@@ -866,18 +857,16 @@ struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd)
   }

   mutex_lock(&sysfs_mutex);
-  spin_lock(&sysfs_assoc_lock);

   /* This, again, can happen if tree structure has
    * changed and we looked up the wrong thing.  Restart.
    */
-  if (cur->s_dentry != dentry) {
+  if (dentry->d_fsdata != cur) {
    dput(dentry);
    sysfs_put(cur);
-   goto restart1;
+   goto restart;
   }

-  spin_unlock(&sysfs_assoc_lock);

   sysfs_put(cur);
  }
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 11/25] sysfs: Remove s_dentry
Posted by ebiederm on Tue, 07 Aug 2007 21:19:31 GMT
View Forum Message <> Reply to Message

The only uses of s_dentry left are the code that maintains
s_dentry and trivial users that don't actually need it.
So this patch removes the s_dentry maintenance code and
restructures the trivial uses to use something else.

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

```
---
 fs/sysfs/dir.c   |   32 ++++----------------------------
 fs/sysfs/mount.c |    1 -
 fs/sysfs/sysfs.h |    1 -
 3 files changed, 4 insertions(+), 30 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index adb1b01..1078e60 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -247,22 +247,7 @@ static void sysfs_d_iput(struct dentry * dentry, struct inode * inode)
 {
  struct sysfs_dirent * sd = dentry->d_fsdata;

- if (sd) {
-  /* sd->s_dentry is protected with sysfs_assoc_lock.
-   * This allows sysfs_drop_dentry() to dereference it.
-   */
-  spin_lock(&sysfs_assoc_lock);
-
-  /* The dentry might have been deleted or another
-   * lookup could have happened updating sd->s_dentry to
-   * point the new dentry.  Ignore if it isn't pointing
-   * to this dentry.
-   */
-  if (sd->s_dentry == dentry)
-   sd->s_dentry = NULL;
-  spin_unlock(&sysfs_assoc_lock);
-  sysfs_put(sd);
- }
+ sysfs_put(sd);
  iput(inode);
 }

@@ -310,9 +295,6 @@ struct sysfs_dirent *sysfs_new_dirent(const char *name, umode_t mode,
int type)
  * @sd: target sysfs_dirent
  * @dentry: dentry to associate
  *
- * Associate @sd with @dentry.  This is protected by
- * sysfs_assoc_lock to avoid race with sysfs_d_iput().
- *
  * LOCKING:
  * mutex_lock(sysfs_mutex)
  */
@@ -320,12 +302,6 @@ static void sysfs_attach_dentry(struct sysfs_dirent *sd, struct dentry
*dentry)
 {
```

```
  dentry->d_op = &sysfs_dentry_ops;
  dentry->d_fsdata = sysfs_get(sd);
-
- /* protect sd->s_dentry against sysfs_d_iput */
- spin_lock(&sysfs_assoc_lock);
- sd->s_dentry = dentry;
- spin_unlock(&sysfs_assoc_lock);
-
  d_rehash(dentry);
 }

@@ -927,7 +903,7 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)

  /* rename */
  d_add(new_dentry, NULL);
- d_move(sd->s_dentry, new_dentry);
+ d_move(old_dentry, new_dentry);

  error = 0;
  goto out_unlock;
@@ -960,7 +936,7 @@ int sysfs_move_dir(struct kobject *kobj, struct kobject *new_parent_kobj)
   error = PTR_ERR(old_dentry);
   goto out_dput;
  }
- old_parent = sd->s_parent->s_dentry;
+ old_parent = old_dentry->d_parent;

  new_parent = sysfs_get_dentry(new_parent_sd);
  if (IS_ERR(new_parent)) {
@@ -986,7 +962,7 @@ again:
  } else
   error = 0;
  d_add(new_dentry, NULL);
- d_move(sd->s_dentry, new_dentry);
+ d_move(old_dentry, new_dentry);
  dput(new_dentry);

  /* Remove from old parent's list and insert into new parent's list. */
diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index 9fae7d5..4968d31 100644
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -57,7 +57,6 @@ static int sysfs_fill_super(struct super_block *sb, void *data, int silent)
   iput(inode);
   return -ENOMEM;
  }
- sysfs_root.s_dentry = root;
  root->d_fsdata = &sysfs_root;
```

```
  sb->s_root = root;
  return 0;
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index 77253aa..8ed13cf 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -37,7 +37,6 @@ struct sysfs_dirent {
  unsigned int  s_flags;
  umode_t   s_mode;
  ino_t   s_ino;
- struct dentry  * s_dentry;
  struct iattr  * s_iattr;
  atomic_t  s_event;
 };
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 12/25] sysfs: Introduce sysfs_rename_mutex

Posted by ebiederm on Tue, 07 Aug 2007 21:21:06 GMT

View Forum Message <> Reply to Message

Looking carefully at the rename code we have a subtle dependency
that the structure of sysfs not change while we are performing
a rename. If the parent directory of the object we are renaming
changes while the rename is being performed nasty things could
happen when we go to release our locks.

So introduce a sysfs_rename_mutex to prevent this highly
unlikely theoretical issue.

In addition hold sysfs_rename_mutex over all calls to
sysfs_get_dentry. Allowing sysfs_get_dentry to be simplified
in the future.

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

```
---
 fs/sysfs/dir.c   |    8 +++++++-
 fs/sysfs/file.c  |    4 ++++
 fs/sysfs/sysfs.h |    1 +
 3 files changed, 12 insertions(+), 1 deletions(-)

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

```
index 1078e60..e0f49b7 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -15,6 +15,7 @@
 #include "sysfs.h"

 DEFINE_MUTEX(sysfs_mutex);
+DEFINE_MUTEX(sysfs_rename_mutex);
 spinlock_t sysfs_assoc_lock = SPIN_LOCK_UNLOCKED;

 static spinlock_t sysfs_ino_lock = SPIN_LOCK_UNLOCKED;
@@ -774,7 +775,7 @@ static struct dentry *__sysfs_get_dentry(struct super_block *sb, struct
sysfs_di
  * down from there looking up dentry for each step.
  *
  * LOCKING:
- * Kernel thread context (may sleep)
+ * mutex_lock(sysfs_rename_mutex)
  *
  * RETURNS:
  * Pointer to found dentry on success, ERR_PTR() value on error.
@@ -859,6 +860,8 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  const char *dup_name = NULL;
  int error;

+ mutex_lock(&sysfs_rename_mutex);
+
  /* get the original dentry */
  sd = kobj->sd;
  old_dentry = sysfs_get_dentry(sd);
@@ -916,6 +919,7 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  kfree(dup_name);
  dput(old_dentry);
  dput(new_dentry);
+ mutex_unlock(&sysfs_rename_mutex);
  return error;
 }

@@ -927,6 +931,7 @@ int sysfs_move_dir(struct kobject *kobj, struct kobject *new_parent_kobj)
  struct dentry *old_dentry = NULL, *new_dentry = NULL;
  int error;

+ mutex_lock(&sysfs_rename_mutex);
  BUG_ON(!sd->s_parent);
  new_parent_sd = new_parent_kobj->sd ? new_parent_kobj->sd : &sysfs_root;

@@ -983,6 +988,7 @@ again:
  dput(new_parent);
```

```
  dput(old_dentry);
  dput(new_dentry);
+ mutex_unlock(&sysfs_rename_mutex);
  return error;
 }

diff --git a/fs/sysfs/file.c b/fs/sysfs/file.c
index 416351a..fe783ea 100644
--- a/fs/sysfs/file.c
+++ b/fs/sysfs/file.c
@@ -470,7 +470,9 @@ int sysfs_update_file(struct kobject * kobj, const struct attribute * attr)
  if (!victim_sd)
   goto out;

+ mutex_lock(&sysfs_rename_mutex);
  victim = sysfs_get_dentry(victim_sd);
+ mutex_unlock(&sysfs_rename_mutex);
  if (IS_ERR(victim)) {
   rc = PTR_ERR(victim);
   victim = NULL;
@@ -509,7 +511,9 @@ int sysfs_chmod_file(struct kobject *kobj, struct attribute *attr, mode_t
mode)
  if (!victim_sd)
   goto out;

+ mutex_lock(&sysfs_rename_mutex);
  victim = sysfs_get_dentry(victim_sd);
+ mutex_unlock(&sysfs_rename_mutex);
  if (IS_ERR(victim)) {
   rc = PTR_ERR(victim);
   victim = NULL;
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index 8ed13cf..791b3ed 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -89,6 +89,7 @@ extern int sysfs_setattr(struct dentry *dentry, struct iattr *iattr);

 extern spinlock_t sysfs_assoc_lock;
 extern struct mutex sysfs_mutex;
+extern struct mutex sysfs_rename_mutex;
 extern struct super_block * sysfs_sb;
 extern const struct file_operations sysfs_dir_operations;
 extern const struct file_operations sysfs_file_operations;
--
1.5.1.1.181.g2de0
```

Containers mailing list

Subject: [PATCH 13/25] sysfs: Simply sysfs_get_dentry
Posted by ebiederm on Tue, 07 Aug 2007 21:22:13 GMT
View Forum Message <> Reply to Message

Now that we know the sysfs tree structure cannot change under us
simplfy sysfs_get_dentry.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/dir.c | 48 ++++++--
 1 files changed, 6 insertions(+), 42 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index e0f49b7..a9bdb12 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -784,71 +784,35 @@ struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd)
 {
  struct sysfs_dirent *cur;
  struct dentry *parent_dentry, *dentry;
- int i, depth;

  /* Find the first parent which has valid dentry.
   */
- mutex_lock(&sysfs_mutex);
- restart:
-
  dentry = NULL;
- depth = 0;
  cur = sd;
  while (!(dentry = __sysfs_get_dentry(sysfs_sb, cur))) {
   if (cur->s_flags & SYSFS_FLAG_REMOVED) {
    dentry = ERR_PTR(-ENOENT);
-   depth = 0;
    break;
   }
   cur = cur->s_parent;
- depth++;
  }

  /* from the found dentry, look up depth times */
- while (depth--) {
- /* find and get depth'th ancestor */
- for (cur = sd, i = 0; cur && i < depth; i++)

```
+ while (dentry->d_fsdata != sd) {
+  /* Find the first ancestor I have not looked up */
+  cur = sd;
+  while (cur->s_parent != dentry->d_fsdata)
    cur = cur->s_parent;

-  /* This can happen if tree structure was modified due
-   * to move/rename.  Restart.
-   */
-  if (i != depth) {
-   dput(dentry);
-   goto restart;
-  }
-
-  sysfs_get(cur);
-
-  mutex_unlock(&sysfs_mutex);
-
   /* look it up */
   parent_dentry = dentry;
   dentry = lookup_one_len_kern(cur->s_name, parent_dentry,
         strlen(cur->s_name));
   dput(parent_dentry);

-  if (IS_ERR(dentry)) {
-   sysfs_put(cur);
-   return dentry;
-  }
-
-  mutex_lock(&sysfs_mutex);
-
-  /* This, again, can happen if tree structure has
-   * changed and we looked up the wrong thing.  Restart.
-   */
-  if (dentry->d_fsdata != cur) {
-   dput(dentry);
-   sysfs_put(cur);
-   goto restart;
-  }
-
-
-  sysfs_put(cur);
+  if (IS_ERR(dentry))
+   break;
  }
-
- mutex_unlock(&sysfs_mutex);
  return dentry;
```

```
 }

--
1.5.1.1.181.g2de0
```

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

---

Subject: [PATCH 14/25] sysfs: Don't use lookup_one_len_kern
Posted by ebiederm on Tue, 07 Aug 2007 21:23:57 GMT
View Forum Message <> Reply to Message

Upon inspection it appears that there is no looking of the
inode mutex in lookup_one_len_kern and we aren't calling
it with the inode mutex and that is wrong.

So this patch rolls our own dcache insertion function that
does exactly what we need it to do. As it turns out this
is pretty trivial to do and it makes the code easier to
audit.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/dir.c | 41 ++++++++++++++++++++++++++++++++++++++---
 1 files changed, 39 insertions(+), 2 deletions(-)

```
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index a9bdb12..1d53c2a 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -765,6 +765,44 @@ static struct dentry *__sysfs_get_dentry(struct super_block *sb, struct
sysfs_di
  return dentry;
 }

+static struct dentry *sysfs_add_dentry(struct dentry *parent, struct sysfs_dirent *sd)
+{
+ struct qstr name;
+ struct dentry *dentry;
+ struct inode *inode;
+
+ mutex_lock(&parent->d_inode->i_mutex);
+ mutex_lock(&sysfs_mutex);
+ dentry = ERR_PTR(-EINVAL);
+ if (parent->d_fsdata != sd->s_parent)
```

```
+  goto out;
+
+ name.name = sd->s_name;
+ name.len = strlen(sd->s_name);
+ dentry = d_hash_and_lookup(parent, &name);
+ if (dentry)
+  goto out;
+
+ dentry = d_alloc(parent, &name);
+ if (!dentry) {
+  dentry = ERR_PTR(-ENOMEM);
+  goto out;
+ }
+
+ inode = sysfs_get_inode(sd);
+ if (!inode) {
+  dput(dentry);
+  dentry = ERR_PTR(-ENOMEM);
+  goto out;
+ }
+ d_instantiate(dentry, inode);
+ sysfs_attach_dentry(sd, dentry);
+out:
+ mutex_unlock(&sysfs_mutex);
+ mutex_unlock(&parent->d_inode->i_mutex);
+ return dentry;
+}
+
 /**
  * sysfs_get_dentry - get dentry for the given sysfs_dirent
  * @sd: sysfs_dirent of interest
@@ -806,8 +844,7 @ struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd)

   /* look it up */
   parent_dentry = dentry;
-  dentry = lookup_one_len_kern(cur->s_name, parent_dentry,
-          strlen(cur->s_name));
+  dentry = sysfs_add_dentry(parent_dentry, cur);
   dput(parent_dentry);

   if (IS_ERR(dentry))
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 15/25] vfs: Remove lookup_one_len_kern
Posted by ebiederm on Tue, 07 Aug 2007 21:25:05 GMT
View Forum Message <> Reply to Message

Now that sysfs no longer uses lookup_one_len_kern the function has
no users so remove it from the kernel. Making namei.c just a little
easier to read.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/namei.c | 46 ++++++++++++----------------------------------
 include/linux/namei.h | 1 -
 2 files changed, 11 insertions(+), 36 deletions(-)

diff --git a/fs/namei.c b/fs/namei.c
index a83160a..69d3304 100644
--- a/fs/namei.c
+++ b/fs/namei.c
@@ -1273,7 +1273,12 @@ int __user_path_lookup_open(const char __user *name, unsigned int
lookup_flags,
  return err;
 }

-static inline struct dentry *__lookup_hash_kern(struct qstr *name, struct dentry *base, struct
nameidata *nd)
+/*
+ * Restricted form of lookup. Doesn't follow links, single-component only,
+ * needs parent already locked. Doesn't follow mounts.
+ * SMP-safe.
+ */
+static inline struct dentry * __lookup_hash(struct qstr *name, struct dentry *base, struct
nameidata *nd)
 {
  struct dentry *dentry;
  struct inode *inode;
@@ -1281,6 +1286,11 @@ static inline struct dentry *__lookup_hash_kern(struct qstr *name,
struct dentry

  inode = base->d_inode;

+ err = permission(inode, MAY_EXEC, nd);
+ dentry = ERR_PTR(err);
+ if (err)
+ goto out;
+
  /*
   * See if the low-level filesystem might want
   * to use its own hash..
@@ -1308,29 +1318,6 @@ out:

```
  return dentry;
 }

-/*
- * Restricted form of lookup. Doesn't follow links, single-component only,
- * needs parent already locked. Doesn't follow mounts.
- * SMP-safe.
- */
-static inline struct dentry * __lookup_hash(struct qstr *name, struct dentry *base, struct
nameidata *nd)
-{
- struct dentry *dentry;
- struct inode *inode;
- int err;
-
- inode = base->d_inode;
-
- err = permission(inode, MAY_EXEC, nd);
- dentry = ERR_PTR(err);
- if (err)
-  goto out;
-
- dentry = __lookup_hash_kern(name, base, nd);
-out:
- return dentry;
-}
-
 static struct dentry *lookup_hash(struct nameidata *nd)
 {
  return __lookup_hash(&nd->last, nd->dentry, nd);
@@ -1369,17 +1356,6 @@ struct dentry *lookup_one_len(const char *name, struct dentry *base,
int len)
  return __lookup_hash(&this, base, NULL);
 }

-struct dentry *lookup_one_len_kern(const char *name, struct dentry *base, int len)
-{
- int err;
- struct qstr this;
-
- err = __lookup_one_len(name, &this, base, len);
- if (err)
-  return ERR_PTR(err);
- return __lookup_hash_kern(&this, base, NULL);
-}
-
 int fastcall __user_walk_fd(int dfd, const char __user *name, unsigned flags,
        struct nameidata *nd)
```

 {
diff --git a/include/linux/namei.h b/include/linux/namei.h
index 6c38efb..36e5690 100644
--- a/include/linux/namei.h
+++ b/include/linux/namei.h
@@ -82,7 +82,6 @@ extern struct file *nameidata_to_filp(struct nameidata *nd, int flags);
 extern void release_open_intent(struct nameidata *);

 extern struct dentry * lookup_one_len(const char *, struct dentry *, int);
-extern struct dentry *lookup_one_len_kern(const char *, struct dentry *, int);

 extern int follow_down(struct vfsmount **, struct dentry **);
 extern int follow_up(struct vfsmount **, struct dentry **);
--
1.5.1.1.181.g2de0

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

---

Subject: [PATCH 16/25] sysfs: Support for preventing unmounts.
Posted by ebiederm on Tue, 07 Aug 2007 21:26:28 GMT
View Forum Message <> Reply to Message

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

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

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

diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index 4968d31..b2bfa45 100644
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -33,47 +33,110 @@ struct sysfs_dirent sysfs_root = {

 static int sysfs_fill_super(struct super_block *sb, void *data, int silent)
 {

```
- struct inode *inode;
- struct dentry *root;
+ struct sysfs_super_info *info = NULL;
+ struct inode *inode = NULL;
+ struct dentry *root = NULL;
+ int error;

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

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

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

```
+ if (sysfs_sb == sb)
+  sysfs_sb = NULL;
+ return error;
 }

 static int sysfs_get_sb(struct file_system_type *fs_type,
  int flags, const char *dev_name, void *data, struct vfsmount *mnt)
 {
- return get_sb_single(fs_type, flags, data, sysfs_fill_super, mnt);
+ int rc;
+ mutex_lock(&sysfs_rename_mutex);
+ rc = get_sb_single(fs_type, flags, data, sysfs_fill_super, mnt);
+ mutex_unlock(&sysfs_rename_mutex);
+ return rc;
 }

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

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

```
+{
+ /* must hold sysfs_rename_mutex */
+ struct super_block *sb;
+restart:
+ spin_lock(&sb_lock);
+ list_for_each_entry(sb, &sysfs_fs_type.fs_supers, s_instances) {
+  if (!sysfs_info(sb)->grabbed)
+   continue;
+  sysfs_info(sb)->grabbed = 0;
+  spin_unlock(&sb_lock);
+  deactivate_super(sb);
+  goto restart;
+ }
+ spin_unlock(&sb_lock);
+}
+
 int __init sysfs_init(void)
 {
  int err = -ENOMEM;
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index 791b3ed..8156ccb 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -50,8 +50,18 @@ struct sysfs_addrm_cxt {
  int   cnt;
 };

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

 extern struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd);
 extern struct sysfs_dirent *sysfs_get_active(struct sysfs_dirent *sd);
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 17/25] sysfs: Rewrite rename in terms of sysfs dirents
Posted by ebiederm on Tue, 07 Aug 2007 21:27:33 GMT
View Forum Message <> Reply to Message

This patch rewrites sysfs_rename_dir to perform it's checks
as much as possible on the underlying sysfs_dirents instead
of the contents of the dcache. It turns out that this version
is a little simpler, and a little more like the rest of
the sysfs directory modification code.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
fs/sysfs/dir.c | 36 ++++++++++++++++++++----------------
1 files changed, 16 insertions(+), 20 deletions(-)

```
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 1d53c2a..3228f5a 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -855,7 +855,7 @@ struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd)

 int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
 {
- struct sysfs_dirent *sd;
+ struct sysfs_dirent *sd = kobj->sd;
  struct dentry *parent = NULL;
  struct dentry *old_dentry = NULL, *new_dentry = NULL;
  const char *dup_name = NULL;
@@ -863,42 +863,41 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)

  mutex_lock(&sysfs_rename_mutex);

+ error = 0;
+ if (strcmp(sd->s_name, new_name) == 0)
+  goto out; /* nothing to rename */
+
  /* get the original dentry */
- sd = kobj->sd;
  old_dentry = sysfs_get_dentry(sd);
  if (IS_ERR(old_dentry)) {
   error = PTR_ERR(old_dentry);
-  goto out_dput;
+  goto out;
  }
```

```
  parent = old_dentry->d_parent;

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

- new_dentry = lookup_one_len(new_name, parent, strlen(new_name));
- if (IS_ERR(new_dentry)) {
-  error = PTR_ERR(new_dentry);
-  goto out_unlock;
- }
-
- error = -EINVAL;
- if (old_dentry == new_dentry)
+ error = -EEXIST;
+ if (sysfs_find_dirent(sd->s_parent, new_name))
    goto out_unlock;

- error = -EEXIST;
- if (new_dentry->d_inode)
+ 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);
  if (!new_name)
-  goto out_drop;
+  goto out_unlock;

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

  mutex_lock(&sysfs_mutex);
  dup_name = sd->s_name;
@@ -910,16 +909,13 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  d_move(old_dentry, new_dentry);

  error = 0;
- goto out_unlock;
-
- out_drop:
- d_drop(new_dentry);
  out_unlock:
```

```
+ mutex_unlock(&sysfs_mutex);
  mutex_unlock(&parent->d_inode->i_mutex);
- out_dput:
  kfree(dup_name);
  dput(old_dentry);
  dput(new_dentry);
+ out:
  mutex_unlock(&sysfs_rename_mutex);
  return error;
 }
--
1.5.1.1.181.g2de0
```


Subject: [PATCH 18/25] sysfs: Rewrite sysfs_move_dir in terms of sysfs dirents
Posted by ebiederm on Tue, 07 Aug 2007 21:28:10 GMT
View Forum Message <> Reply to Message

This patch rewrites sysfs_move_dir to perform it's checks
as much as possible on the underlying sysfs_dirents instead
of the contents of the dcache, making sysfs_move_dir
more like the rest of the sysfs directory modification
code.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/dir.c | 34 ++++++++++++++++++++----------
 1 files changed, 18 insertions(+), 16 deletions(-)

```
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 3228f5a..94d705a 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -932,56 +932,58 @@ int sysfs_move_dir(struct kobject *kobj, struct kobject
*new_parent_kobj)
  BUG_ON(!sd->s_parent);
  new_parent_sd = new_parent_kobj->sd ? new_parent_kobj->sd : &sysfs_root;

+ error = 0;
+ if (sd->s_parent == new_parent_sd)
+  goto out; /* nothing to move */
+
  /* get dentries */
```

```
  old_dentry = sysfs_get_dentry(sd);
  if (IS_ERR(old_dentry)) {
   error = PTR_ERR(old_dentry);
-  goto out_dput;
+  goto out;
  }
  old_parent = old_dentry->d_parent;

  new_parent = sysfs_get_dentry(new_parent_sd);
  if (IS_ERR(new_parent)) {
   error = PTR_ERR(new_parent);
-  goto out_dput;
+  goto out;
  }

- if (old_parent->d_inode == new_parent->d_inode) {
-  error = 0;
-  goto out_dput; /* nothing to move */
- }
 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);
   goto again;
  }
+ mutex_lock(&sysfs_mutex);

- new_dentry = lookup_one_len(kobj->name, new_parent, strlen(kobj->name));
- if (IS_ERR(new_dentry)) {
-  error = PTR_ERR(new_dentry);
+ error = -EEXIST;
+ if (sysfs_find_dirent(new_parent_sd, sd->s_name))
   goto out_unlock;
- } else
-  error = 0;
+
+ error = -ENOMEM;
+ new_dentry = d_alloc_name(new_parent, sd->s_name);
+ if (!new_dentry)
+  goto out_unlock;
+
+ error = 0;
  d_add(new_dentry, NULL);
  d_move(old_dentry, new_dentry);
  dput(new_dentry);

  /* Remove from old parent's list and insert into new parent's list. */
- mutex_lock(&sysfs_mutex);
```

```
-
  sysfs_unlink_sibling(sd);
  sysfs_get(new_parent_sd);
  sysfs_put(sd->s_parent);
  sd->s_parent = new_parent_sd;
  sysfs_link_sibling(sd);

- mutex_unlock(&sysfs_mutex);
-
  out_unlock:
+ mutex_unlock(&sysfs_mutex);
  mutex_unlock(&new_parent->d_inode->i_mutex);
  mutex_unlock(&old_parent->d_inode->i_mutex);
- out_dput:
+ out:
  dput(new_parent);
  dput(old_dentry);
  dput(new_dentry);
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 19/25] sysfs: sysfs_get_dentry add a sb parameter
Posted by ebiederm on Tue, 07 Aug 2007 21:29:23 GMT
View Forum Message <> Reply to Message

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

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

```
 fs/sysfs/dir.c   |   12 +++++++-----
 fs/sysfs/file.c  |    4 ++--
 fs/sysfs/sysfs.h |    2 +-
 3 files changed, 10 insertions(+), 8 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 94d705a..ac45523 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -805,6 +805,7 @@ out:

 /**
```

```
  * 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
@@ -817,8 +818,9 @@ out:
  *
  * RETURNS:
  * Pointer to found dentry on success, ERR_PTR() value on error.
+ * NULL if the sysfs dentry does not appear in the specified superblock
  */
-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 *cur;
  struct dentry *parent_dentry, *dentry;
@@ -827,7 +829,7 @@ struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd)
   */
  dentry = NULL;
  cur = sd;
- while (!(dentry = __sysfs_get_dentry(sysfs_sb, cur))) {
+ while (!(dentry = __sysfs_get_dentry(sb, cur))) {
   if (cur->s_flags & SYSFS_FLAG_REMOVED) {
    dentry = ERR_PTR(-ENOENT);
    break;
@@ -868,7 +870,7 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
   goto out; /* nothing to rename */

  /* get the original dentry */
- old_dentry = sysfs_get_dentry(sd);
+ old_dentry = sysfs_get_dentry(sysfs_sb, sd);
  if (IS_ERR(old_dentry)) {
   error = PTR_ERR(old_dentry);
   goto out;
@@ -937,14 +939,14 @@ int sysfs_move_dir(struct kobject *kobj, struct kobject
*new_parent_kobj)
   goto out; /* nothing to move */

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

- new_parent = sysfs_get_dentry(new_parent_sd);
```

```
+ new_parent = sysfs_get_dentry(sysfs_sb, new_parent_sd);
  if (IS_ERR(new_parent)) {
   error = PTR_ERR(new_parent);
   goto out;
diff --git a/fs/sysfs/file.c b/fs/sysfs/file.c
index fe783ea..f954b9f 100644
--- a/fs/sysfs/file.c
+++ b/fs/sysfs/file.c
@@ -471,7 +471,7 @@ int sysfs_update_file(struct kobject * kobj, const struct attribute * attr)
   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);
@@ -512,7 +512,7 @@ int sysfs_chmod_file(struct kobject *kobj, struct attribute *attr, mode_t
mode)
   goto out;

  mutex_lock(&sysfs_rename_mutex);
- victim = sysfs_get_dentry(victim_sd);
+ victim = sysfs_get_dentry(sysfs_sb, victim_sd);
  mutex_unlock(&sysfs_rename_mutex);
  if (IS_ERR(victim)) {
   rc = PTR_ERR(victim);
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index 8156ccb..6de7e2b 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -63,7 +63,7 @@ extern struct file_system_type sysfs_fs_type;
 void sysfs_grab_supers(void);
 void sysfs_release_supers(void);

-extern struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd);
+extern struct dentry *sysfs_get_dentry(struct super_block *sb, struct sysfs_dirent *sd);
 extern struct sysfs_dirent *sysfs_get_active(struct sysfs_dirent *sd);
 extern void sysfs_put_active(struct sysfs_dirent *sd);
 extern struct sysfs_dirent *sysfs_get_active_two(struct sysfs_dirent *sd);
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 20/25] sysfs: Rename Support multiple superblocks
Posted by ebiederm on Tue, 07 Aug 2007 21:31:18 GMT
View Forum Message <> Reply to Message

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

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

```
---
 fs/sysfs/dir.c |  193 ++++++++++++++++++++++++++++++++++++++++++------------------------
 1 files changed, 136 insertions(+), 57 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index ac45523..34eabf4 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -855,42 +855,113 @@ struct dentry *sysfs_get_dentry(struct super_block *sb, struct
sysfs_dirent *sd)
  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)
+   continue;
+  if (IS_ERR(dentry)) {
+   error = PTR_ERR(dentry);
+   goto err_out;
+  }
+
+  srs = kzalloc(sizeof(*srs), GFP_KERNEL);
+  if (!srs) {
+   dput(dentry);
+   goto err_out;
+  }
+
+  INIT_LIST_HEAD(&srs->list);
+  list_add(head, &srs->list);
+  srs->old_dentry = dentry;
+  srs->old_parent = dget(dentry->d_parent);
+
+  dentry = sysfs_get_dentry(sb, new_parent_sd);
+  if (IS_ERR(dentry)) {
+   error = PTR_ERR(dentry);
+   goto err_out;
+  }
+  srs->new_parent = dentry;
+
+  error = -ENOMEM;
+  dentry = d_alloc_name(srs->new_parent, name);
+  if (!dentry)
+   goto err_out;
+  srs->new_dentry = dentry;
+ }
+ return 0;
+
+err_out:
+ post_rename(head);
+ return error;
+}
+
+
 int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
 {
  struct sysfs_dirent *sd = kobj->sd;
```

```
- struct dentry *parent = NULL;
- struct dentry *old_dentry = NULL, *new_dentry = NULL;
+ struct list_head todo;
+ struct sysfs_rename_struct *srs;
+ struct inode *parent_inode = NULL;
  const char *dup_name = NULL;
  int error;

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

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

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

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

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

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

- error = -ENOMEM;
- new_dentry = d_alloc_name(parent, new_name);
- if (!new_dentry)
-  goto out_unlock;
-
```

```
  /* rename kobject and sysfs_dirent */
  error = -ENOMEM;
  new_name = dup_name = kstrdup(new_name, GFP_KERNEL);
@@ -901,23 +972,25 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  if (error)
   goto out_unlock;

- mutex_lock(&sysfs_mutex);
  dup_name = sd->s_name;
  sd->s_name = new_name;
- mutex_unlock(&sysfs_mutex);

  /* 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;
 }
@@ -926,10 +999,12 @@ int sysfs_move_dir(struct kobject *kobj, struct kobject
*new_parent_kobj)
 {
  struct sysfs_dirent *sd = kobj->sd;
  struct sysfs_dirent *new_parent_sd;
- struct dentry *old_parent, *new_parent = NULL;
- struct dentry *old_dentry = NULL, *new_dentry = NULL;
+ struct list_head todo;
+ struct sysfs_rename_struct *srs;
+ struct inode *old_parent_inode = NULL, *new_parent_inode = NULL;
```

```
  int error;

+ INIT_LIST_HEAD(&todo);
  mutex_lock(&sysfs_rename_mutex);
  BUG_ON(!sd->s_parent);
  new_parent_sd = new_parent_kobj->sd ? new_parent_kobj->sd : &sysfs_root;
@@ -938,24 +1013,29 @@ int sysfs_move_dir(struct kobject *kobj, struct kobject
*new_parent_kobj)
  if (sd->s_parent == new_parent_sd)
   goto out; /* nothing to move */

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

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

 again:
- mutex_lock(&old_parent->d_inode->i_mutex);
- if (!mutex_trylock(&new_parent->d_inode->i_mutex)) {
-  mutex_unlock(&old_parent->d_inode->i_mutex);
+ mutex_lock(&old_parent_inode->i_mutex);
+ if (!mutex_trylock(&new_parent_inode->i_mutex)) {
```

```
+  mutex_unlock(&old_parent_inode->i_mutex);
   goto again;
  }
  mutex_lock(&sysfs_mutex);
@@ -964,15 +1044,11 @@ again:
  if (sysfs_find_dirent(new_parent_sd, sd->s_name))
   goto out_unlock;

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

  /* Remove from old parent's list and insert into new parent's list. */
  sysfs_unlink_sibling(sd);
@@ -981,14 +1057,17 @@ again:
  sd->s_parent = new_parent_sd;
  sysfs_link_sibling(sd);

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

 }
--
1.5.1.1.181.g2de0

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

---

Subject: [PATCH 21/25] sysfs: sysfs_chmod_file handle multiple superblocks
Posted by ebiederm on Tue, 07 Aug 2007 21:32:46 GMT

View Forum Message <> Reply to Message

Teach sysfs_chmod_file how to handle multiple sysfs
superblocks. We need to iterate over each superblock
so that we give all of the appropriate filesystem modification
notifications.

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

```
---
 fs/sysfs/file.c |   41 +++++++++++++++++++++++++----------------
 1 files changed, 25 insertions(+), 16 deletions(-)

diff --git a/fs/sysfs/file.c b/fs/sysfs/file.c
index f954b9f..cff054f 100644
--- a/fs/sysfs/file.c
+++ b/fs/sysfs/file.c
@@ -501,7 +501,8 @@ int sysfs_update_file(struct kobject * kobj, const struct attribute * attr)
 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;
@@ -512,22 +513,30 @@ int sysfs_chmod_file(struct kobject *kobj, struct attribute *attr, mode_t mode)
   goto out;

  mutex_lock(&sysfs_rename_mutex);
- victim = sysfs_get_dentry(sysfs_sb, victim_sd);
- mutex_unlock(&sysfs_rename_mutex);
- if (IS_ERR(victim)) {
-  rc = PTR_ERR(victim);
-  victim = NULL;
```

```
-  goto out;
+ sysfs_grab_supers();
+ list_for_each_entry(sb, &sysfs_fs_type.fs_supers, s_instances) {
+  victim = sysfs_get_dentry(sb, victim_sd);
+  if (!victim)
+   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);
+  mutex_unlock(&inode->i_mutex);
+
+  dput(victim);
  }
-
- 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);
- mutex_unlock(&inode->i_mutex);
- out:
- dput(victim);
+out_unlock:
+ sysfs_release_supers();
+ mutex_unlock(&sysfs_rename_mutex);
+out:
  sysfs_put(victim_sd);
  return rc;
 }
--
1.5.1.1.181.g2de0
```

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

---

Subject: [PATCH 22/25] sysfs: sysfs_uptdate_file handle multiple superblocks

Teach sysfs_update_file how to handle multiple sysfs
superblocks. Again we are just iterating over the superblocks
to so all of the filesystem modification notifications work
as expected.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
fs/sysfs/file.c | 35 ++++++++++++++++++++++++++++++++---------
1 files changed, 21 insertions(+), 14 deletions(-)

```
diff --git a/fs/sysfs/file.c b/fs/sysfs/file.c
index cff054f..1e6f9df 100644
--- a/fs/sysfs/file.c
+++ b/fs/sysfs/file.c
@@ -462,7 +462,8 @@ EXPORT_SYMBOL_GPL(sysfs_add_file_to_group);
 int sysfs_update_file(struct kobject * kobj, const struct attribute * attr)
 {
  struct sysfs_dirent *victim_sd = NULL;
- struct dentry *victim = NULL;
+ struct super_block *sb;
+ struct dentry *victim;
  int rc;

  rc = -ENOENT;
@@ -471,21 +472,27 @@ int sysfs_update_file(struct kobject * kobj, const struct attribute * attr)
   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;
+ sysfs_grab_supers();
+ list_for_each_entry(sb, &sysfs_fs_type.fs_supers, s_instances) {
+  victim = sysfs_get_dentry(sb, victim_sd);
+  if (!victim)
+   continue;
+  if (IS_ERR(victim)) {
+   rc = PTR_ERR(victim);
+   victim = NULL;
+   goto out_unlock;
+  }
+  mutex_lock(&victim->d_inode->i_mutex);
+  victim->d_inode->i_mtime = CURRENT_TIME;
```

```
+  fsnotify_modify(victim);
+  mutex_unlock(&victim->d_inode->i_mutex);
+  rc = 0;
+  dput(victim);
  }
-
- mutex_lock(&victim->d_inode->i_mutex);
- victim->d_inode->i_mtime = CURRENT_TIME;
- fsnotify_modify(victim);
- mutex_unlock(&victim->d_inode->i_mutex);
- rc = 0;
+ out_unlock:
+ sysfs_release_supers();
+ mutex_unlock(&sysfs_rename_mutex);
  out:
- dput(victim);
  sysfs_put(victim_sd);
  return rc;
 }
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 23/25] sysfs: Implement sysfs tagged directory support.
Posted by ebiederm on Tue, 07 Aug 2007 21:35:31 GMT
View Forum Message <> Reply to Message

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

What this patch does is to add an additional tag field to the
sysfs dirent structure. For directories that should show different
contents depending on the context such as /sys/class/net/, and
/sys/devices/virtual/net/ this tag field is used to specify the
context in which those directories should be visible. Effectively
this is the same as creating multiple distinct directories with
the same name the internally to sysfs the result is nicer.

I am calling the concept of a single directory that looks like multiple
directories all at the same path in the filesystem tagged directories.

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

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 uniteresting problem
to solve.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/bin.c | 2 +-
 fs/sysfs/dir.c | 180 ++----
 fs/sysfs/file.c | 8 +-
 fs/sysfs/group.c | 12 ++--
 fs/sysfs/inode.c | 6 +-
 fs/sysfs/mount.c | 44 ++++++++++++++-
 fs/sysfs/symlink.c | 2 +-
 fs/sysfs/sysfs.h | 15 ++++-
 include/linux/sysfs.h | 17 +++++
 9 files changed, 253 insertions(+), 33 deletions(-)

diff --git a/fs/sysfs/bin.c b/fs/sysfs/bin.c
index 135353f..1ef0a07 100644

```
--- a/fs/sysfs/bin.c
+++ b/fs/sysfs/bin.c
@@ -248,7 +248,7 @@ int sysfs_create_bin_file(struct kobject * kobj, struct bin_attribute * attr)

 void sysfs_remove_bin_file(struct kobject * kobj, struct bin_attribute * attr)
 {
- if (sysfs_hash_and_remove(kobj->sd, attr->attr.name) < 0) {
+ if (sysfs_hash_and_remove(kobj, kobj->sd, attr->attr.name) < 0) {
    printk(KERN_ERR "%s: "
     "bad dentry or inode or no such file: \"%s\"\n",
     __FUNCTION__, attr->attr.name);
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 34eabf4..bc30a8a 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -387,10 +387,16 @@ void sysfs_addrm_start(struct sysfs_addrm_cxt *acxt,
  */
 int sysfs_add_one(struct sysfs_addrm_cxt *acxt, struct sysfs_dirent *sd)
 {
- if (sysfs_find_dirent(acxt->parent_sd, sd->s_name))
+ const void *tag = NULL;
+
+ tag = sysfs_creation_tag(acxt->parent_sd, sd);
+
+ if (sysfs_find_dirent(acxt->parent_sd, tag, sd->s_name))
    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);
@@ -540,13 +546,18 @@ void sysfs_addrm_finish(struct sysfs_addrm_cxt *acxt)
  * Pointer to sysfs_dirent if found, NULL if not.
  */
 struct sysfs_dirent *sysfs_find_dirent(struct sysfs_dirent *parent_sd,
+           const void *tag,
           const unsigned char *name)
 {
  struct sysfs_dirent *sd;

- for (sd = parent_sd->s_children; sd; sd = sd->s_sibling)
+ for (sd = parent_sd->s_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;
 }

@@ -570,7 +581,7 @@ struct sysfs_dirent *sysfs_get_dirent(struct sysfs_dirent *parent_sd,
  struct sysfs_dirent *sd;

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

@@ -636,13 +647,16 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry
*dentry,
     struct nameidata *nd)
 {
  struct dentry *ret = NULL;
- struct sysfs_dirent *parent_sd = dentry->d_parent->d_fsdata;
+ struct dentry *parent = dentry->d_parent;
+ struct sysfs_dirent *parent_sd = parent->d_fsdata;
  struct sysfs_dirent *sd;
  struct inode *inode;
+ const void *tag;

  mutex_lock(&sysfs_mutex);

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

  /* no such entry */
  if (!sd)
@@ -825,6 +839,14 @@ struct dentry *sysfs_get_dentry(struct super_block *sb, struct
sysfs_dirent *sd)
  struct sysfs_dirent *cur;
  struct dentry *parent_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 NULL;
+ }
+
  /* Find the first parent which has valid dentry.
```

```
   */
  dentry = NULL;
@@ -926,7 +948,6 @@ err_out:
  return error;
 }

-
 int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
 {
  struct sysfs_dirent *sd = kobj->sd;
@@ -934,19 +955,24 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  struct sysfs_rename_struct *srs;
  struct inode *parent_inode = NULL;
  const char *dup_name = NULL;
+ const void *old_tag, *tag;
  int error;

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

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

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

  error = -ENOMEM;
  mutex_lock(&sysfs_mutex);
@@ -959,7 +985,7 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  mutex_lock(&sysfs_mutex);

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

  /* rename kobject and sysfs_dirent */
```

```
@@ -974,6 +1000,8 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)

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

  /* rename */
  list_for_each_entry(srs, &todo, list) {
@@ -981,6 +1009,20 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
   d_move(srs->old_dentry, srs->new_dentry);
  }

+ /* If we are moving across superblocks drop the dcache entries */
+ if (old_tag != tag) {
+  struct super_block *sb;
+  struct dentry *dentry;
+  list_for_each_entry(sb, &sysfs_fs_type.fs_supers, s_instances) {
+   dentry = __sysfs_get_dentry(sb, sd);
+   if (!dentry)
+    continue;
+   shrink_dcache_parent(dentry);
+   d_drop(dentry);
+   dput(dentry);
+  }
+ }
+
  error = 0;
 out_unlock:
  mutex_unlock(&sysfs_mutex);
@@ -1003,11 +1045,13 @@ int sysfs_move_dir(struct kobject *kobj, struct kobject
*new_parent_kobj)
  struct sysfs_rename_struct *srs;
  struct inode *old_parent_inode = NULL, *new_parent_inode = NULL;
  int error;
+ const void *tag;

  INIT_LIST_HEAD(&todo);
  mutex_lock(&sysfs_rename_mutex);
  BUG_ON(!sd->s_parent);
  new_parent_sd = new_parent_kobj->sd ? new_parent_kobj->sd : &sysfs_root;
+ tag = sysfs_dirent_tag(sd);

  error = 0;
  if (sd->s_parent == new_parent_sd)
@@ -1041,7 +1085,7 @@ 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;
@@ -1080,10 +1124,11 @@ static inline unsigned char dt_type(struct sysfs_dirent *sd)

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

  if (filp->f_pos == 0) {
   ino = parent_sd->s_ino;
@@ -1101,6 +1146,8 @@ static int sysfs_readdir(struct file * filp, void * dirent, filldir_t filldir)
  if ((filp->f_pos > 1) && (filp->f_pos < UINT_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_children;
   while (pos && (filp->f_pos > pos->s_ino))
@@ -1110,6 +1157,10 @@ static int sysfs_readdir(struct file * filp, void * dirent, filldir_t filldir)
    const char * name;
    int len;

+   if ((parent_sd->s_flags & SYSFS_FLAG_TAGGED) &&
+       (pos->s_tag.tag != tag))
+    continue;
+
    name = pos->s_name;
    len = strlen(name);
    filp->f_pos = ino = pos->s_ino;
@@ -1130,3 +1181,104 @@ const struct file_operations sysfs_dir_operations = {
  .read  = generic_read_dir,
  .readdir = sysfs_readdir,
 };
+
+const void *sysfs_creation_tag(struct sysfs_dirent *parent_sd, struct sysfs_dirent *sd)
+{
+ const void *tag = NULL;
```

```
+
+ if (parent_sd->s_flags & SYSFS_FLAG_TAGGED) {
+  struct kobject *kobj;
+  switch (sysfs_type(sd)) {
+  case SYSFS_DIR:
+   kobj = sd->s_elem.dir.kobj;
+   break;
+  case SYSFS_KOBJ_LINK:
+   kobj = sd->s_elem.symlink.target_sd->s_elem.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_children && (sysfs_type(sd) == SYSFS_DIR) &&
+     !(sd->s_flags & SYSFS_FLAG_REMOVED) &&
+     !(sd->s_flags & SYSFS_FLAG_TAGGED) &&
+     sd->s_parent &&
+     !(sd->s_parent->s_flags & SYSFS_FLAG_TAGGED)) {
+  err = 0;
+  sd->s_flags |= SYSFS_FLAG_TAGGED;
+  sd->s_tag.ops = tag_ops;
+ }
+ mutex_unlock(&sysfs_mutex);
+ return err;
```

```
+}
diff --git a/fs/sysfs/file.c b/fs/sysfs/file.c
index 1e6f9df..11a385c 100644
--- a/fs/sysfs/file.c
+++ b/fs/sysfs/file.c
@@ -369,9 +369,9 @@ void sysfs_notify(struct kobject *k, char *dir, char *attr)
  mutex_lock(&sysfs_mutex);

  if (sd && dir)
-  sd = sysfs_find_dirent(sd, dir);
+  sd = sysfs_find_dirent(sd, NULL, dir);
  if (sd && attr)
-  sd = sysfs_find_dirent(sd, attr);
+  sd = sysfs_find_dirent(sd, NULL, attr);
  if (sd) {
   atomic_inc(&sd->s_event);
   wake_up_interruptible(&k->poll);
@@ -560,7 +560,7 @@ 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);
 }


@@ -577,7 +577,7 @@ void sysfs_remove_file_from_group(struct kobject *kobj,

  dir_sd = sysfs_get_dirent(kobj->sd, group);
  if (dir_sd) {
-  sysfs_hash_and_remove(dir_sd, attr->name);
+  sysfs_hash_and_remove(kobj, dir_sd, attr->name);
   sysfs_put(dir_sd);
  }
 }
diff --git a/fs/sysfs/group.c b/fs/sysfs/group.c
index 4606f7c..9e928fd 100644
--- a/fs/sysfs/group.c
+++ b/fs/sysfs/group.c
@@ -17,16 +17,16 @@
 #include "sysfs.h"


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

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

-static int create_files(struct sysfs_dirent *dir_sd,
+static int create_files(struct kobject *kobj, struct sysfs_dirent *dir_sd,
     const struct attribute_group *grp)
 {
  struct attribute *const* attr;
@@ -35,7 +35,7 @@ static int create_files(struct sysfs_dirent *dir_sd,
  for (attr = grp->attrs; *attr && !error; attr++)
   error = sysfs_add_file(dir_sd, *attr, SYSFS_KOBJ_ATTR);
  if (error)
-  remove_files(dir_sd, grp);
+  remove_files(kobj, dir_sd, grp);
  return error;
 }

@@ -55,7 +55,7 @@ int sysfs_create_group(struct kobject * kobj,
  } else
   sd = kobj->sd;
  sysfs_get(sd);
- error = create_files(sd, grp);
+ error = create_files(kobj, sd, grp);
  if (error) {
   if (grp->name)
    sysfs_remove_subdir(sd);
@@ -76,7 +76,7 @@ void sysfs_remove_group(struct kobject * kobj,
  } else
   sd = sysfs_get(dir_sd);

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

diff --git a/fs/sysfs/inode.c b/fs/sysfs/inode.c
index 70a2420..d870efd 100644
--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -212,17 +212,19 @@ struct inode * sysfs_get_inode(struct sysfs_dirent *sd)
  return inode;
 }

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

```
 {
  struct sysfs_addrm_cxt acxt;
  struct sysfs_dirent *sd;
+ const void *tag;

  if (!dir_sd)
   return -ENOENT;

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

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

diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index b2bfa45..7668e66 100644
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -67,6 +67,7 @@ static int sysfs_fill_super(struct super_block *sb, void *data, int silent)
   goto out_err;
  }
  root->d_fsdata = &sysfs_root;
+ root->d_sb = sb;
  sb->s_root = root;
  sb->s_fs_info = info;
  return 0;
@@ -80,20 +81,55 @@ out_err:
  return error;
 }

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

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

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

 void sysfs_grab_supers(void)
diff --git a/fs/sysfs/symlink.c b/fs/sysfs/symlink.c
index 46f8fd4..99aaf6f 100644
--- a/fs/sysfs/symlink.c
+++ b/fs/sysfs/symlink.c
@@ -113,7 +113,7 @@ int sysfs_create_link(struct kobject * kobj, struct kobject * target, const
char

 void sysfs_remove_link(struct kobject * kobj, const char * name)
```

```
 {
- sysfs_hash_and_remove(kobj->sd, name);
+ sysfs_hash_and_remove(kobj, kobj->sd, name);
 }

 static int sysfs_get_target_path(struct sysfs_dirent * parent_sd,
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index 6de7e2b..832d675 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -28,6 +28,10 @@ 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  dir;
   struct sysfs_elem_symlink symlink;
   struct sysfs_elem_attr  attr;
@@ -52,6 +56,7 @@ 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)
@@ -63,6 +68,13 @@ extern struct file_system_type sysfs_fs_type;
 void sysfs_grab_supers(void);
 void sysfs_release_supers(void);

+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);
 extern struct dentry *sysfs_get_dentry(struct super_block *sb, struct sysfs_dirent *sd);
 extern struct sysfs_dirent *sysfs_get_active(struct sysfs_dirent *sd);
 extern void sysfs_put_active(struct sysfs_dirent *sd);
@@ -80,6 +92,7 @@ extern struct inode * sysfs_get_inode(struct sysfs_dirent *sd);

 extern void release_sysfs_dirent(struct sysfs_dirent * sd);
 extern struct sysfs_dirent *sysfs_find_dirent(struct sysfs_dirent *parent_sd,
+          const void *tag,
```

```
            const unsigned char *name);
 extern struct sysfs_dirent *sysfs_get_dirent(struct sysfs_dirent *parent_sd,
           const unsigned char *name);
@@ -88,7 +101,7 @@ extern struct sysfs_dirent *sysfs_new_dirent(const char *name, umode_t
mode,

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

 extern int sysfs_create_subdir(struct kobject *kobj, const char *name,
diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
index c16e4c5..28c497e 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -74,6 +74,14 @@ 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);
+};
+
 #define SYSFS_TYPE_MASK  0x00ff
 #define SYSFS_ROOT  0x0001
 #define SYSFS_DIR  0x0002
@@ -84,6 +92,7 @@ struct sysfs_ops {

 #define SYSFS_FLAG_MASK  ~SYSFS_TYPE_MASK
 #define SYSFS_FLAG_REMOVED 0x0100
+#define SYSFS_FLAG_TAGGED 0x0200

 #ifdef CONFIG_SYSFS

@@ -134,6 +143,8 @@ void sysfs_remove_file_from_group(struct kobject *kobj,

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

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

```
 #else /* CONFIG_SYSFS */
@@ -229,6 +240,12 @@ static inline void sysfs_notify(struct kobject * k, char *dir, char *attr)
 {
 }

+static inline int sysfs_enable_tagging(struct kobject *kobj,
+    const struct sysfs_tagged_dir_operations *tag_ops)
+{
+ return 0;
+}
+
 static inline int __must_check sysfs_init(void)
 {
  return 0;
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 24/25] sysfs: Implement sysfs_delete_link and sysfs_rename_link
Posted by ebiederm on Tue, 07 Aug 2007 21:36:09 GMT
View Forum Message <> Reply to Message

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

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

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

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

```
---
 fs/sysfs/symlink.c    |   31 +++++++++++++++++++++++++++++++
 include/linux/sysfs.h |   18 ++++++++++++++++++
```

```
 2 files changed, 49 insertions(+), 0 deletions(-)

diff --git a/fs/sysfs/symlink.c b/fs/sysfs/symlink.c
index 99aaf6f..6476b8f 100644
--- a/fs/sysfs/symlink.c
+++ b/fs/sysfs/symlink.c
@@ -106,6 +106,21 @@ int sysfs_create_link(struct kobject * kobj, struct kobject * target, const
char


 /**
+ * sysfs_delete_link - remove symlink in object's directory.
+ * @kobj: object we're acting for.
+ * @targ: object we're pointing to.
+ * @name: name of the symlink to remove.
+ *
+ * Unlike sysfs_remove_link sysfs_delete_link has enough information
+ * to successfully delete symlinks in tagged directories.
+ */
+void sysfs_delete_link(struct kobject *kobj, struct kobject *targ,
+   const char *name)
+{
+ sysfs_hash_and_remove(targ, kobj->sd, name);
+}
+
+/**
  * sysfs_remove_link - remove symlink in object's directory.
  * @kobj: object we're acting for.
  * @name: name of the symlink to remove.
@@ -116,6 +131,22 @@ void sysfs_remove_link(struct kobject * kobj, const char * name)
  sysfs_hash_and_remove(kobj, kobj->sd, name);
 }

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

```
+
 static int sysfs_get_target_path(struct sysfs_dirent * parent_sd,
      struct sysfs_dirent * target_sd, char *path)
 {
diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
index 28c497e..90eb35e 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -129,6 +129,13 @@ sysfs_create_link(struct kobject * kobj, struct kobject * target, const char
* n
 extern void
 sysfs_remove_link(struct kobject *, const char * name);

+extern int
+sysfs_rename_link(struct kobject *kobj, struct kobject *target,
+    const char *old_name, const char *new_name);
+
+extern void
+sysfs_delete_link(struct kobject *dir, struct kobject *targ, const char *name);
+
 int __must_check sysfs_create_bin_file(struct kobject *kobj,
      struct bin_attribute *attr);
 void sysfs_remove_bin_file(struct kobject *kobj, struct bin_attribute *attr);
@@ -204,6 +211,17 @@ static inline void sysfs_remove_link(struct kobject * k, const char *
name)
  ;
 }

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

 static inline int sysfs_create_bin_file(struct kobject * k, struct bin_attribute * a)
 {
--
1.5.1.1.181.g2de0
```

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

Subject: [PATCH 25/25] driver core: Implement tagged directory support for device classes.
Posted by ebiederm on Tue, 07 Aug 2007 21:36:42 GMT
View Forum Message <> Reply to Message

This patch enables tagging on every class directory if struct class
has tag_ops.

In addition device_del and device_rename were modified to use
sysfs_delete_link and sysfs_rename_link respectively to ensure
when these operations happen on devices whos classes have
tag_ops that they work properly.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/base/class.c | 30 ++++++++++++++++++++++++++----
 drivers/base/core.c | 45 ++++++++++++++++++++++++++++++---------------
 include/linux/device.h | 2 ++
 3 files changed, 52 insertions(+), 25 deletions(-)

diff --git a/drivers/base/class.c b/drivers/base/class.c
index 4d22226..1485e2a 100644
--- a/drivers/base/class.c
+++ b/drivers/base/class.c
@@ -134,6 +134,17 @@ static void remove_class_attrs(struct class * cls)
  }
 }

+static int class_setup_tagging(struct class *cls)
+{
+ const struct sysfs_tagged_dir_operations *tag_ops;
+
+ tag_ops = cls->tag_ops;
+ if (!tag_ops)
+ return 0;
+
+ return sysfs_enable_tagging(&cls->subsys.kobj, tag_ops);
+}
+
 int class_register(struct class * cls)
 {
  int error;
@@ -152,11 +163,22 @@ int class_register(struct class * cls)
  subsys_set_kset(cls, class_subsys);

```
  error = subsystem_register(&cls->subsys);
- if (!error) {
-  error = add_class_attrs(class_get(cls));
-  class_put(cls);
- }
+ if (error)
+  goto out;
+
+ error = class_setup_tagging(cls);
+ if (error)
+  goto out_unregister;
+
+ error = add_class_attrs(cls);
+ if (error)
+  goto out_unregister;
+
+out:
  return error;
+out_unregister:
+ subsystem_unregister(&cls->subsys);
+ goto out;
 }

 void class_unregister(struct class * cls)
diff --git a/drivers/base/core.c b/drivers/base/core.c
index e6738bc..8663d96 100644
--- a/drivers/base/core.c
+++ b/drivers/base/core.c
@@ -637,8 +637,14 @@ static struct kobject * get_device_parent(struct device *dev,
    return kobj;

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

  if (parent)
@@ -934,8 +940,8 @@ void device_del(struct device * dev)
   /* If this is not a "fake" compatible device, remove the
    * symlink from the class to the device. */
   if (dev->kobj.parent != &dev->class->subsys.kobj)
```

```
-   sysfs_remove_link(&dev->class->subsys.kobj,
-       dev->bus_id);
+   sysfs_delete_link(&dev->class->subsys.kobj,
+       &dev->kobj, dev->bus_id);
    if (parent) {
 #ifdef CONFIG_SYSFS_DEPRECATED
     char *class_name = make_class_name(dev->class->name,
@@ -1233,6 +1239,13 @@ int device_rename(struct device *dev, char *new_name)
  strlcpy(old_device_name, dev->bus_id, BUS_ID_SIZE);
  strlcpy(dev->bus_id, new_name, BUS_ID_SIZE);

+ if (dev->class && (dev->kobj.parent != &dev->class->subsys.kobj)) {
+  error = sysfs_rename_link(&dev->class->subsys.kobj,
+    &dev->kobj, old_device_name, new_name);
+  if (error)
+   goto out;
+ }
+
  error = kobject_rename(&dev->kobj, new_name);
  if (error) {
   strlcpy(dev->bus_id, old_device_name, BUS_ID_SIZE);
@@ -1241,27 +1254,17 @@ int device_rename(struct device *dev, char *new_name)

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

- if (dev->class) {
-  sysfs_remove_link(&dev->class->subsys.kobj, old_device_name);
-  error = sysfs_create_link(&dev->class->subsys.kobj, &dev->kobj,
-        dev->bus_id);
-  if (error) {
-   /* Uh... how to unravel this if restoring can fail? */
-   dev_err(dev, "%s: sysfs_create_symlink failed (%d)\n",
-     __FUNCTION__, error);
-  }
+  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);
+ }
+#endif
```

```
+  error = sysfs_rename_link(&dev->parent->kobj, &dev->kobj,
+        old_class_name, new_class_name);
+  if (error)
+   goto out;
  }
+#endif
 out:
  put_device(dev);

diff --git a/include/linux/device.h b/include/linux/device.h
index 3a38d1f..4b90c39 100644
--- a/include/linux/device.h
+++ b/include/linux/device.h
@@ -200,6 +200,8 @@ struct class {

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

 extern int __must_check class_register(struct class *);
--
1.5.1.1.181.g2de0
```

Subject: Re: [PATCH 01/25] sysfs: Move all of inode initialization into sysfs_init_inode

Posted by Tejun Heo on Wed, 08 Aug 2007 06:37:14 GMT

View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:08:07PM -0600, Eric W. Biederman wrote:
>
> Signed-off-by: "Eric W. Biederman" <ebiederm@xmission.com>

Acked-by: Tejun Heo <htejun@gmail.com>

Subject: Re: [PATCH 02/25] sysfs: Remove sysfs_instantiate
Posted by Tejun Heo on Wed, 08 Aug 2007 06:37:42 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:08:50PM -0600, Eric W. Biederman wrote:
>
> Now that sysfs_get_inode is dropping the inode lock
> we no longer have a need from sysfs_instantiate.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Acked-by: Tejun Heo <htejun@gmail.com>

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

Subject: Re: [PATCH 03/25] sysfs: Use kill_anon_super
Posted by Tejun Heo on Wed, 08 Aug 2007 06:50:39 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:10:27PM -0600, Eric W. Biederman wrote:
>
> Since sysfs no longer stores fs directory information in the dcache
> on a permanent basis kill_litter_super it is inappropriate and actively
> wrong. It will decrement the count on all dentries left in the
> dcache before trying to free them.
>
> At the moment this is not biting us only because we never unmount sysfs.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Acked-by: Tejun Heo <htejun@gmail.com>

--
tejun

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

Subject: Re: [PATCH 04/25] sysfs: Make sysfs_mount static
Posted by Tejun Heo on Wed, 08 Aug 2007 06:51:03 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:11:16PM -0600, Eric W. Biederman wrote:

>
> This patch modifies the users of sysfs_mount to use sysfs_root
> instead (which is what they are looking for). It then
> makes sysfs_mount static to keep people from using it
> by accident.
>
> The net result is slightly faster and cleaner code.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Acked-by: Tejun Heo <htejun@gmail.com>

--
tejun

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

---

Subject: Re: [PATCH 05/25] sysfs: In sysfs_lookup don't open code sysfs_find_dirent
Posted by Tejun Heo on Wed, 08 Aug 2007 06:51:23 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:12:02PM -0600, Eric W. Biederman wrote:
>
> This is a small cleanup patch that makes the code just
> a little bit cleaner.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Acked-by: Tejun Heo <htejun@gmail.com>

--
tejun

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

---

Subject: Re: [PATCH 06/25] sysfs: Simplify readdir.
Posted by Tejun Heo on Wed, 08 Aug 2007 07:12:41 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:13:14PM -0600, Eric W. Biederman wrote:

>
> At some point someone wrote sysfs_readdir to insert a cursor
> into the list of sysfs_dirents to ensure that sysfs_readdir would
> restart properly. That works but it is complex code and tends
> to be expensive.
>
> The same effect can be achieved by keeping the sysfs_dirents in
> inode order and using the inode number as the f_pos. Then
> when we restart we just have to find the first dirent whose inode
> number is equal or greater then the last sysfs_dirent we attempted
> to return.
>
> Removing the sysfs directory cursor also allows the remove of
> all of the mysterious checks for sysfs_type(sd) != 0.   Which
> were nonbovious checks to see if a cursor was in a directory list.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Acked-by: Tejun Heo <htejun@gmail.com>

--
tejun

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

---

Subject: Re: [PATCH 07/25] sysfs: Rewrite sysfs_drop_dentry.
Posted by Tejun Heo on Wed, 08 Aug 2007 07:35:55 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:14:56PM -0600, Eric W. Biederman wrote:
>
> Currently we find the dentry to drop by looking at sd->s_dentry.
> We can just as easily accomplish the same task by looking up the
> sysfs inode and finding all of the dentries from there, with the
> added bonus that we don't need to play with the sysfs_assoc_lock.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Great, Acked-by: Tejun Heo <htejun@gmail.com>

--
tejun

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

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by Cornelia Huck on Wed, 08 Aug 2007 07:38:15 GMT
View Forum Message <> Reply to Message

On Tue, 07 Aug 2007 15:06:21 -0600,
ebiederm@xmission.com (Eric W. Biederman) wrote:

>
> The following patchset applies on top of the last round of
> sysfs cleanups that Tejun sent out on the 2nd.
>
> My target with this patchset is to support sysfs directories
> with a tag on struct sysfs_dirent making them visible only
> on selected mounts of sysfs.
>
> After going around and around the different possibilities I
> believe I have finally found something that works and is
> reasonably maintainable. I believe I have achieved that
> with only introducing some extra complexity in a few very localized
> places.
>
> The worst part is the code to support multiple superblocks and thus
> multiple dentry tress for sysfs. I had allocate a linked list in
> sysfs_move_dir for all of the possible dentries I would need to call
> d_move on. Bleh. It works, it is correct and it is an atomic
> rename.

My udev failed to create /dev/dasd* so it cannot mount root :(I'm
currently trying to find out what causes this, may take some time...

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

Subject: Re: [PATCH 08/25] sysfs: Implement __sysfs_get_dentry
Posted by Tejun Heo on Wed, 08 Aug 2007 07:45:20 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:16:19PM -0600, Eric W. Biederman wrote:
>
> This function is similar but much simpler to sysfs_get_dentry
> returns a sysfs dentry if one currently exists.
>

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

Acked-by: Tejun Heo <htejun@gmail.com>

--
tejun

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

---

Subject: Re: [PATCH 09/25] sysfs: Move sysfs_get_dentry below __sysfs_get_dentry
Posted by Tejun Heo on Wed, 08 Aug 2007 07:45:39 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:17:33PM -0600, Eric W. Biederman wrote:
>
> sysfs_get_dentry is higher in fs/sysfs/dir.c then is needed and it the
> dependencies get simpler if we move it down in the file to where I
> have placed __sysfs_get_dentry. So this patch just moves
> sysfs_get_dentry so code movement doesn't get confused with later code
> changes.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Acked-by: Tejun Heo <htejun@gmail.com>

--
tejun

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

---

Subject: Re: [PATCH 10/25] sysfs: Rewrite sysfs_get_dentry in terms of __sysfs_get_dentry
Posted by Tejun Heo on Wed, 08 Aug 2007 07:45:59 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:18:27PM -0600, Eric W. Biederman wrote:
>
> This removes the last major user of s_dentry and makes
> the locking in sysfs_get_dentry much simpler. Hopefully
> leading to more readable and maintainable code.

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

Acked-by: Tejun Heo <htejun@gmail.com>

--
tejun

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

---

Subject: Re: [PATCH 11/25] sysfs: Remove s_dentry
Posted by Tejun Heo on Wed, 08 Aug 2007 07:46:37 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:19:31PM -0600, Eric W. Biederman wrote:
>
> The only uses of s_dentry left are the code that maintains
> s_dentry and trivial users that don't actually need it.
> So this patch removes the s_dentry maintenance code and
> restructures the trivial uses to use something else.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Acked-by: Tejun Heo <htejun@gmail.com>

Nice clean up. Thanks.

--
tejun

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

---

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by ebiederm on Wed, 08 Aug 2007 07:47:51 GMT
View Forum Message <> Reply to Message

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

> My udev failed to create /dev/dasd* so it cannot mount root :(I'm
> currently trying to find out what causes this, may take some time...

Oh weird.

No great surprise that something goofed up given how many
patches were involved. Still there shouldn't have been
any user visible differences in the patchset.

If you can narrow down which patch caused the problem that would be
great.

Otherwise I guess I will have to start looking at the code and
trying to guess what might have triggered this.

Eric

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

---

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by Tejun Heo on Wed, 08 Aug 2007 07:53:23 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Cornelia Huck <cornelia.huck@de.ibm.com> writes:
>
>> My udev failed to create /dev/dasd* so it cannot mount root :(I'm
>> currently trying to find out what causes this, may take some time...
>
> Oh weird.
>
> No great surprise that something goofed up given how many
> patches were involved. Still there shouldn't have been
> any user visible differences in the patchset.
>
> If you can narrow down which patch caused the problem that would be
> great.
>
> Otherwise I guess I will have to start looking at the code and
> trying to guess what might have triggered this.

JFYI, your patchset works fine on my x86-64 test machine. Corenelia
tests on a s390 system which does rename/move stuff with its devices and
usually gets hit first if something is wrong. I'll scream if I find
anything suspicious while reviewing.

Thanks.

--
tejun

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

---

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by Cornelia Huck on Wed, 08 Aug 2007 07:54:32 GMT
View Forum Message <> Reply to Message

On Wed, 08 Aug 2007 01:47:51 -0600,
ebiederm@xmission.com (Eric W. Biederman) wrote:

> Cornelia Huck <cornelia.huck@de.ibm.com> writes:
>
> > My udev failed to create /dev/dasd* so it cannot mount root :(I'm
> > currently trying to find out what causes this, may take some time...
>
> Oh weird.
>
> No great surprise that something goofed up given how many
> patches were involved. Still there shouldn't have been
> any user visible differences in the patchset.
>
> If you can narrow down which patch caused the problem that would be
> great.

Got it: It's patch 6, the readdir simplification.

(The udev on that guest is ancient (063)...)

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

---

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by ebiederm on Wed, 08 Aug 2007 07:57:07 GMT
View Forum Message <> Reply to Message

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

> On Wed, 08 Aug 2007 01:47:51 -0600,
> ebiederm@xmission.com (Eric W. Biederman) wrote:

>
>> Cornelia Huck <cornelia.huck@de.ibm.com> writes:
>>
>> > My udev failed to create /dev/dasd* so it cannot mount root :(I'm
>> > currently trying to find out what causes this, may take some time...
>>
>> Oh weird.
>>
>> No great surprise that something goofed up given how many
>> patches were involved. Still there shouldn't have been
>> any user visible differences in the patchset.
>>
>> If you can narrow down which patch caused the problem that would be
>> great.
>
> Got it: It's patch 6, the readdir simplification.
>
> (The udev on that guest is ancient (063)...)

Ok. That is weird.

Does it depend on the order in which the dentries are returned from
readdir?

Unless I made a really stupid error otherwise the two versions
of readdir should have the same semantics.

Eric

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

---

Subject: Re: [PATCH 12/25] sysfs: Introduce sysfs_rename_mutex
Posted by Tejun Heo on Wed, 08 Aug 2007 08:19:32 GMT
View Forum Message <> Reply to Message

Hello, Eric.

Eric W. Biederman wrote:
> Looking carefully at the rename code we have a subtle dependency
> that the structure of sysfs not change while we are performing
> a rename. If the parent directory of the object we are renaming
> changes while the rename is being performed nasty things could
> happen when we go to release our locks.
>
> So introduce a sysfs_rename_mutex to prevent this highly

> unlikely theoretical issue.

Yeah, it's a theoretical issue. Rename/move implementation has always
depended on the parent structure not changing beneath it, but it's nice
to tighten up loose ends.

> +DEFINE_MUTEX(sysfs_rename_mutex);

Probably doesn't really matter but wouldn't a rwsem fit better?

> @@ -774,7 +775,7 @@ static struct dentry *__sysfs_get_dentry(struct super_block *sb, struct
sysfs_di
>   * down from there looking up dentry for each step.
>   *
>   * LOCKING:
> - * Kernel thread context (may sleep)
> + * mutex_lock(sysfs_rename_mutex)

LOCKING describes what locks should be held when entering the function,
so proper description would be something like...

 Kernel thread context, grabs sysfs_rename_mutex

Thanks.

--
tejun

Subject: Re: [PATCH 12/25] sysfs: Introduce sysfs_rename_mutex
Posted by Tejun Heo on Wed, 08 Aug 2007 08:23:11 GMT
View Forum Message <> Reply to Message

Tejun Heo wrote:
> Hello, Eric.
>
> Eric W. Biederman wrote:
>> Looking carefully at the rename code we have a subtle dependency
>> that the structure of sysfs not change while we are performing
>> a rename. If the parent directory of the object we are renaming
>> changes while the rename is being performed nasty things could
>> happen when we go to release our locks.
>>
>> So introduce a sysfs_rename_mutex to prevent this highly

>> unlikely theoretical issue.
>
> Yeah, it's a theoretical issue. Rename/move implementation has always
> depended on the parent structure not changing beneath it, but it's nice
> to tighten up loose ends.
>
>> +DEFINE_MUTEX(sysfs_rename_mutex);
>
> Probably doesn't really matter but wouldn't a rwsem fit better?
>
>> @@ -774,7 +775,7 @@ static struct dentry *__sysfs_get_dentry(struct super_block *sb, struct
sysfs_di
>>   * down from there looking up dentry for each step.
>>   *
>>   * LOCKING:
>> - * Kernel thread context (may sleep)
>> + * mutex_lock(sysfs_rename_mutex)
>
> LOCKING describes what locks should be held when entering the function,
> so proper description would be something like...
>
> Kernel thread context, grabs sysfs_rename_mutex

Oops, forget about the above. Thought the comment was added to
sysfs_rename_dir().

--
tejun

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

---

Subject: Re: [PATCH 13/25] sysfs: Simply sysfs_get_dentry
Posted by Tejun Heo on Wed, 08 Aug 2007 08:24:29 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:22:13PM -0600, Eric W. Biederman wrote:
>
> Now that we know the sysfs tree structure cannot change under us
> simplfy sysfs_get_dentry.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Acked-by: Tejun Heo <htejun@gmail.com>

It might be better to have sysfs_get_dentry_locked() and

sysfs_get_dentry() so that the latter can be used from
sysfs_update/chmod_file().

--
tejun

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

---

Subject: Re: [PATCH 12/25] sysfs: Introduce sysfs_rename_mutex
Posted by ebiederm on Wed, 08 Aug 2007 08:28:52 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> Hello, Eric.
>
> Eric W. Biederman wrote:
>> Looking carefully at the rename code we have a subtle dependency
>> that the structure of sysfs not change while we are performing
>> a rename. If the parent directory of the object we are renaming
>> changes while the rename is being performed nasty things could
>> happen when we go to release our locks.
>>
>> So introduce a sysfs_rename_mutex to prevent this highly
>> unlikely theoretical issue.
>
> Yeah, it's a theoretical issue. Rename/move implementation has always
> depended on the parent structure not changing beneath it, but it's nice
> to tighten up loose ends.
>
>> +DEFINE_MUTEX(sysfs_rename_mutex);
>
> Probably doesn't really matter but wouldn't a rwsem fit better?

Maybe. I didn't feel any loss in when I was writing the code.
Very few code paths actually seem to care.

>> @@ -774,7 +775,7 @@ static struct dentry *__sysfs_get_dentry(struct
> super_block *sb, struct sysfs_di
>>   * down from there looking up dentry for each step.
>>   *
>>   * LOCKING:
>> - * Kernel thread context (may sleep)
>> + * mutex_lock(sysfs_rename_mutex)

Well this is weird in that it should be on sysfs_get_dentry
more then __sysfs_get_dentry but otherwise it's ok.

> LOCKING describes what locks should be held when entering the function,
> so proper description would be something like...
>
> Kernel thread context, grabs sysfs_rename_mutex

For rename_dir and move_dir yes. I was updating the rules
for sysfs_get_dentry. Which really wants it's parents to
hold that lock.

Eric

_____

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by Cornelia Huck on Wed, 08 Aug 2007 08:37:59 GMT
View Forum Message <> Reply to Message

On Wed, 08 Aug 2007 01:57:07 -0600,
ebiederm@xmission.com (Eric W. Biederman) wrote:

> > Got it: It's patch 6, the readdir simplification.
> >
> > (The udev on that guest is ancient (063)...)
>
> Ok. That is weird.

More weirdness. If I activate another dasd from the repair file
system, /dev/dasdb is created...

Same if I set the card reader online: /dev/vmrdr-0.0.000c is created as
expected.

> Does it depend on the order in which the dentries are returned from
> readdir?

I'd think not.

> Unless I made a really stupid error otherwise the two versions
> of readdir should have the same semantics.

Yes, your patch looks sane. I have no idea why it breaks stuff...

_____

Subject: Re: [PATCH 14/25] sysfs: Don't use lookup_one_len_kern
Posted by Tejun Heo on Wed, 08 Aug 2007 08:38:15 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:23:57PM -0600, Eric W. Biederman wrote:
>
> Upon inspection it appears that there is no looking of the
> inode mutex in lookup_one_len_kern and we aren't calling
> it with the inode mutex and that is wrong.
>
> So this patch rolls our own dcache insertion function that
> does exactly what we need it to do. As it turns out this
> is pretty trivial to do and it makes the code easier to
> audit.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
> ---
> fs/sysfs/dir.c | 41 ++++++++++++++++++++++++++++++++++++++---
> 1 files changed, 39 insertions(+), 2 deletions(-)
>
> diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
> index a9bdb12..1d53c2a 100644
> --- a/fs/sysfs/dir.c
> +++ b/fs/sysfs/dir.c
> @@ -765,6 +765,44 @@ static struct dentry *__sysfs_get_dentry(struct super_block *sb, struct
sysfs_di
>   return dentry;
> }
>
> +static struct dentry *sysfs_add_dentry(struct dentry *parent, struct sysfs_dirent *sd)
> +{
> + struct qstr name;
> + struct dentry *dentry;
> + struct inode *inode;
> +
> + mutex_lock(&parent->d_inode->i_mutex);
> + mutex_lock(&sysfs_mutex);
> + dentry = ERR_PTR(-EINVAL);
> + if (parent->d_fsdata != sd->s_parent)
> + goto out;
> +
> + name.name = sd->s_name;
> + name.len = strlen(sd->s_name);

> + dentry = d_hash_and_lookup(parent, &name);
> + if (dentry)
> + goto out;
> +
> + dentry = d_alloc(parent, &name);
> + if (!dentry) {
> + dentry = ERR_PTR(-ENOMEM);
> + goto out;
> + }
> +
> + inode = sysfs_get_inode(sd);
> + if (!inode) {
> + dput(dentry);
> + dentry = ERR_PTR(-ENOMEM);
> + goto out;
> + }
> + d_instantiate(dentry, inode);
> + sysfs_attach_dentry(sd, dentry);
> +out:
> + mutex_unlock(&sysfs_mutex);
> + mutex_unlock(&parent->d_inode->i_mutex);
> + return dentry;
> +}

This is virtually identical to

```
 mutex_lock(&parent_dentry->d_inode->i_mutex);
 dentry = lookup_one_len_kern(cur->s_name, parent_dentry,
        strlen(cur->s_name));
 mutex_unlock(&parent_dentry->d_inode->i_mutex);
```

right? I don't think we need to duplicate the code here. Or is it
needed for later multi-sb thing?

--
tejun

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

---

Subject: Re: [PATCH 15/25] vfs: Remove lookup_one_len_kern
Posted by Tejun Heo on Wed, 08 Aug 2007 08:39:53 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:25:05PM -0600, Eric W. Biederman wrote:
>

> Now that sysfs no longer uses lookup_one_len_kern the function has
> no users so remove it from the kernel. Making namei.c just a little
> easier to read.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Oh.. you're killing lookup_one_len_kern(). In that case, I think the
previous one is okay too.

Acked-by: Tejun Heo <htejun@gmail.com>

--
tejun

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

---

Subject: Re: [PATCH 17/25] sysfs: Rewrite rename in terms of sysfs dirents
Posted by Tejun Heo on Wed, 08 Aug 2007 08:51:33 GMT
View Forum Message <> Reply to Message

> int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
> {
> - struct sysfs_dirent *sd;
> + struct sysfs_dirent *sd = kobj->sd;
>   struct dentry *parent = NULL;
>   struct dentry *old_dentry = NULL, *new_dentry = NULL;
>   const char *dup_name = NULL;
> @@ -863,42 +863,41 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
>
>   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(sd);
>   if (IS_ERR(old_dentry)) {
>   error = PTR_ERR(old_dentry);
> + goto out;
>   }
>
>   parent = old_dentry->d_parent;
>
>   /* lock parent and get dentry for new name */

```
>   mutex_lock(&parent->d_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);
>   if (!new_name)
> +  goto out_unlock;
>
>   error = kobject_set_name(kobj, "%s", new_name);
>   if (error)
> +  goto out_unlock;
>
>   mutex_lock(&sysfs_mutex);
```

sysfs_mutex is being grabbed twice and unlocked twice later.

--
tejun

Subject: Re: [PATCH 18/25] sysfs: Rewrite sysfs_move_dir in terms of sysfs dirents
Posted by Tejun Heo on Wed, 08 Aug 2007 08:53:04 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:28:10PM -0600, Eric W. Biederman wrote:
>
> This patch rewrites sysfs_move_dir to perform it's checks
> as much as possible on the underlying sysfs_dirents instead
> of the contents of the dcache, making sysfs_move_dir
> more like the rest of the sysfs directory modification
> code.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Signed-off-by: Tejun Heo <htejun@gmail.com>

--
tejun

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

---

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by Tejun Heo on Wed, 08 Aug 2007 08:54:44 GMT
View Forum Message <> Reply to Message

Cornelia Huck wrote:
> On Wed, 08 Aug 2007 01:57:07 -0600,
> ebiederm@xmission.com (Eric W. Biederman) wrote:
>
>>> Got it: It's patch 6, the readdir simplification.
>>>
>>> (The udev on that guest is ancient (063)...)
>> Ok. That is weird.
>
> More weirdness. If I activate another dasd from the repair file
> system, /dev/dasdb is created...
>
> Same if I set the card reader online: /dev/vmrdr-0.0.000c is created as
> expected.
>
>> Does it depend on the order in which the dentries are returned from
>> readdir?
>
> I'd think not.
>
>> Unless I made a really stupid error otherwise the two versions
>> of readdir should have the same semantics.
>
> Yes, your patch looks sane. I have no idea why it breaks stuff...

Are you sure it's patch 6? Patch 17 adds a deadlock in rename path.

--
tejun

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

Subject: Re: [PATCH 19/25] sysfs: sysfs_get_dentry add a sb parameter
Posted by Tejun Heo on Wed, 08 Aug 2007 08:57:03 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:29:23PM -0600, Eric W. Biederman wrote:
>
> In preparation for multiple mounts of sysfs add a superblock parameter to
> sysfs_get_dentry.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
> @@ -827,7 +829,7 @@ struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd)
> */
> dentry = NULL;
> cur = sd;
> - while (!(dentry = __sysfs_get_dentry(sysfs_sb, cur))) {
> + while (!(dentry = __sysfs_get_dentry(sb, cur))) {

It's probably better to add @sb to __sysfs_get_dentry() here too.
That will make things look clearer.

Thanks.

--
tejun

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

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by Cornelia Huck on Wed, 08 Aug 2007 09:16:00 GMT
View Forum Message <> Reply to Message

On Wed, 08 Aug 2007 17:54:44 +0900,
Tejun Heo <htejun@gmail.com> wrote:

> Are you sure it's patch 6? Patch 17 adds a deadlock in rename path.

Yes. If I apply the series up to patch 5, everything's fine. If I apply
patch 6, /dev/dasda* is missing.

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

Subject: Re: [PATCH 20/25] sysfs: Rename Support multiple superblocks
Posted by Tejun Heo on Wed, 08 Aug 2007 09:35:05 GMT
View Forum Message <> Reply to Message

Hello,

On Tue, Aug 07, 2007 at 03:31:18PM -0600, Eric W. Biederman wrote:
> This patch modifies the sysfs_rename_dir and sysfs_move_dir
> to support multiple sysfs dentry trees rooted in different
> sysfs superblocks.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

> +struct sysfs_rename_struct {
> + struct list_head list;
> + struct dentry *old_dentry;
> + struct dentry *new_dentry;
> + struct dentry *old_parent;
> + struct dentry *new_parent;
> +};

Please rename to sysfs_rename_cxt to it consistent with
sysfs_addrm_cxt.

> +static void post_rename(struct list_head *head)

Please rename to sysfs_post_rename() and add comment.

> +{
> + 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)

Ditto.

> +{
> + 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)
> +   continue;

sysfs_get_dentry() return ERR_PTR() value. Oops, sysfs_get_dentry()
implementation is wrong too. Also, please move
sysfs_grab/release_supers() near this patch and add (a lot of)
comments there.

Other than that, I think this is as clean as this can be. Great.

--
tejun

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

---

Subject: Re: [PATCH 21/25] sysfs: sysfs_chmod_file handle multiple superblocks
Posted by Tejun Heo on Wed, 08 Aug 2007 09:38:30 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:32:46PM -0600, Eric W. Biederman wrote:
>
> Teach sysfs_chmod_file how to handle multiple sysfs
> superblocks. We need to iterate over each superblock
> so that we give all of the appropriate filesystem modification
> notifications.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
> ---
> fs/sysfs/file.c | 41 ++++++++++++++++++++++++++++++++---------
> 1 files changed, 25 insertions(+), 16 deletions(-)
>
> diff --git a/fs/sysfs/file.c b/fs/sysfs/file.c
> index f954b9f..cff054f 100644
> --- a/fs/sysfs/file.c
> +++ b/fs/sysfs/file.c
> @@ -501,7 +501,8 @@ int sysfs_update_file(struct kobject * kobj, const struct attribute * attr)
> 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;
> @@ -512,22 +513,30 @@ int sysfs_chmod_file(struct kobject *kobj, struct attribute *attr,
mode_t mode)
>   goto out;
>
>   mutex_lock(&sysfs_rename_mutex);
> - victim = sysfs_get_dentry(sysfs_sb, victim_sd);
> - mutex_unlock(&sysfs_rename_mutex);
> - if (IS_ERR(victim)) {
> - rc = PTR_ERR(victim);
> - victim = NULL;
> - goto out;
> + sysfs_grab_supers();
> + list_for_each_entry(sb, &sysfs_fs_type.fs_supers, s_instances) {
> + victim = sysfs_get_dentry(sb, victim_sd);
> + if (!victim)
> +   continue;
> + if (IS_ERR(victim)) {
> +   rc = PTR_ERR(victim);
> +   victim = NULL;
> +   goto out_unlock;

Hmmm... Please fix sysfs_get_dentry() and make it return either NULL
or PTR_ERR() values. Returning both is pretty confusing. Also, it
would be nice if we can use the rename_prep stuff for this too but it
might just be a wishful thinking.

Thanks.

--
tejun

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

---

Subject: Re: [PATCH 22/25] sysfs: sysfs_uptdate_file handle multiple superblocks
Posted by Tejun Heo on Wed, 08 Aug 2007 09:39:09 GMT
View Forum Message <> Reply to Message

On Tue, Aug 07, 2007 at 03:34:11PM -0600, Eric W. Biederman wrote:
>

> Teach sysfs_update_file how to handle multiple sysfs
> superblocks. Again we are just iterating over the superblocks
> to so all of the filesystem modification notifications work
> as expected.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Same as the previous one.

--
tejun

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

---

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by Cornelia Huck on Wed, 08 Aug 2007 14:16:31 GMT
View Forum Message <> Reply to Message

On Wed, 8 Aug 2007 10:37:59 +0200,
Cornelia Huck <cornelia.huck@de.ibm.com> wrote:

> On Wed, 08 Aug 2007 01:57:07 -0600,
> ebiederm@xmission.com (Eric W. Biederman) wrote:
>
> > > Got it: It's patch 6, the readdir simplification.
> > >
> > > (The udev on that guest is ancient (063)...)
> >
> > Ok. That is weird.
>
> More weirdness. If I activate another dasd from the repair file
> system, /dev/dasdb is created...
>
> Same if I set the card reader online: /dev/vmrdr-0.0.000c is created as
> expected.

OK, it seems that it is udevstart that has problems. (Normal udev
operation seems fine; when I trigger uevents manually, the device nodes
are created.)

I ran strace on udevstart and found that it skipped
/sys/block/dasd/dasda1 (interestingly, /dev/dasda was created after I
ran udevstart manually; no idea why this didn't happen on boot. Further
manual runs don't create /dev/dasda1, however.) ls doesn't produce
different output on the two kernels.

Here is some excerpt from the strace run with the broken kernel:

```
open("/sys/block", O_RDONLY|O_NONBLOCK|O_LARGEFILE|O_DIRECTORY) = 3
fstat64(3, {st_mode=S_IFDIR|0755, st_size=0, ...}) = 0
fcntl64(3, F_SETFD, FD_CLOEXEC)         = 0
getdents64(3, /* 28 entries */, 4096)   = 800
_llseek(3, 3414, [3414], SEEK_SET)      = 0
stat64("/sys/block/dasda/dev", {st_mode=S_IFREG|0444, st_size=4096, ...}) = 0
open("/sys/block/dasda", O_RDONLY|O_NONBLOCK|O_LARGEFILE|O_DIRECTORY) = 4
fstat64(4, {st_mode=S_IFDIR|0755, st_size=0, ...}) = 0
fcntl64(4, F_SETFD, FD_CLOEXEC)         = 0
getdents64(4, /* 15 entries */, 4096)   = 440
_llseek(4, 3079, [3079], SEEK_SET)      = 0
stat64("/sys/block/dasda/uevent/dev", 0x7fb8c3f8) = -1 ENOTDIR (Not a directory)
stat64("/sys/block/dasda/dev/dev", 0x7fb8c3f8) = -1 ENOTDIR (Not a directory)
stat64("/sys/block/dasda/range/dev", 0x7fb8c3f8) = -1 ENOTDIR (Not a directory)
stat64("/sys/block/dasda/removable/dev", 0x7fb8c3f8) = -1 ENOTDIR (Not a directory)
stat64("/sys/block/dasda/size/dev", 0x7fb8c3f8) = -1 ENOTDIR (Not a directory)
stat64("/sys/block/dasda/stat/dev", 0x7fb8c3f8) = -1 ENOTDIR (Not a directory)
stat64("/sys/block/dasda/capability/dev", 0x7fb8c3f8) = -1 ENOTDIR (Not a directory)
stat64("/sys/block/dasda/device/dev", 0x7fb8c3f8) = -1 ENOENT (No such file or d irectory)
stat64("/sys/block/dasda/subsystem/dev", 0x7fb8c3f8) = -1 ENOENT (No such file or directory)
stat64("/sys/block/dasda/holders/dev", 0x7fb8c3f8) = -1 ENOENT (No such file or directory)
stat64("/sys/block/dasda/slaves/dev", 0x7fb8c3f8) = -1 ENOENT (No such file or directory)
stat64("/sys/block/dasda/queue/dev", 0x7fb8c3f8) = -1 ENOENT (No such file or directory)
getdents64(4, /* 1 entries */, 4096)    = 32
close(4)                                = 0
getdents64(3, /* 1 entries */, 4096)    = 32
close(3)                                = 0
lstat64("/sys/block/dasda", {st_mode=S_IFDIR|0755, st_size=0, ...}) = 0
lstat64("/sys/block/dasda/dev", {st_mode=S_IFREG|0444, st_size=4096, ...}) = 0
stat64("/sys/block/dasda/dev", {st_mode=S_IFREG|0444, st_size=4096, ...}) = 0
open("/sys/block/dasda/dev", O_RDONLY)  = 3
read(3, "94:0\n", 4096)                 = 5
close(3)                                = 0
lstat64("/sys/block/dasda/device", {st_mode=S_IFLNK|0777, st_size=0, ...}) = 0
readlink("/sys/block/dasda/device", "../../devices/css0/0.0.0003/0.0.4e30", 256) = 36
lstat64("/sys/devices/css0/0.0.0003/0.0.4e30", {st_mode=S_IFDIR|0755, st_size=0, ...}) = 0
lstat64("/sys/devices/css0/0.0.0003/0.0.4e30/bus", {st_mode=S_IFLNK|0777, st_siz e=0, ...}) = 0
readlink("/sys/devices/css0/0.0.0003/0.0.4e30/bus", "../../../../bus/ccw", 256) = 19
lstat64("/sys/devices/css0/0.0.0003/0.0.4e30/driver", {st_mode=S_IFLNK|0777, st_ size=0, ...}) =
0
readlink("/sys/devices/css0/0.0.0003/0.0.4e30/driver", "../../../../bus/ccw/driv ers/dasd-eckd", 256)
= 37
open("/proc/filesystems", O_RDONLY|O_LARGEFILE) = 3
read(3, "nodev\tsysfs\nnodev\trootfs\nnodev\tb"..., 4095) = 259
close(3)                                = 0
```

```
stat64("/dev/dasda", 0x7fb8b738)        = -1 ENOENT (No such file or directory)
mknod("/dev/dasda", S_IFBLK|0640, makedev(94, 0)) = 0
chmod("/dev/dasda", 060640)             = 0
chown("/dev/dasda", 0, 6)               = 0
```

And here's with a working kernel:

```
open("/sys/block", O_RDONLY|O_NONBLOCK|O_LARGEFILE|O_DIRECTORY) = 3
fstat64(3, {st_mode=S_IFDIR|0755, st_size=0, ...}) = 0
fcntl64(3, F_SETFD, FD_CLOEXEC)         = 0
getdents64(3, /* 27 entries */, 4096)   = 768
stat64("/sys/block/dasda/dev", {st_mode=S_IFREG|0444, st_size=4096, ...}) = 0
open("/sys/block/dasda", O_RDONLY|O_NONBLOCK|O_LARGEFILE|O_DIRECTORY) = 4
fstat64(4, {st_mode=S_IFDIR|0755, st_size=0, ...}) = 0
fcntl64(4, F_SETFD, FD_CLOEXEC)         = 0
getdents64(4, /* 15 entries */, 4096)   = 440
stat64("/sys/block/dasda/dasda1/dev", {st_mode=S_IFREG|0444, st_size=4096, ...}) = 0
stat64("/sys/block/dasda/queue/dev", 0x7f9004f8) = -1 ENOENT (No such file or directory)
stat64("/sys/block/dasda/slaves/dev", 0x7f9004f8) = -1 ENOENT (No such file or directory)
stat64("/sys/block/dasda/holders/dev", 0x7f9004f8) = -1 ENOENT (No such file or directory)
stat64("/sys/block/dasda/subsystem/dev", 0x7f9004f8) = -1 ENOENT (No such file or directory)
stat64("/sys/block/dasda/device/dev", 0x7f9004f8) = -1 ENOENT (No such file or directory)
stat64("/sys/block/dasda/capability/dev", 0x7f9004f8) = -1 ENOTDIR (Not a directory)
stat64("/sys/block/dasda/stat/dev", 0x7f9004f8) = -1 ENOTDIR (Not a  directory)
stat64("/sys/block/dasda/size/dev", 0x7f9004f8) = -1 ENOTDIR (Not a directory)
stat64("/sys/block/dasda/removable/dev", 0x7f9004f8) = -1 ENOTDIR (Not a directory)
stat64("/sys/block/dasda/range/dev", 0x7f9004f8) = -1 ENOTDIR (Not a directory)
stat64("/sys/block/dasda/dev/dev", 0x7f9004f8) = -1 ENOTDIR (Not a directory)
stat64("/sys/block/dasda/uevent/dev", 0x7f9004f8) = -1 ENOTDIR (Not a directory)
getdents64(4, /* 0 entries */, 4096)    = 0
close(4)                                = 0
...
lstat64("/sys/block/dasda", {st_mode=S_IFDIR|0755, st_size=0, ...}) = 0
lstat64("/sys/block/dasda/dev", {st_mode=S_IFREG|0444, st_size=4096, ...}) = 0
stat64("/sys/block/dasda/dev", {st_mode=S_IFREG|0444, st_size=4096, ...}) = 0
open("/sys/block/dasda/dev", O_RDONLY)  = 3
read(3, "94:0\n", 4096)                 = 5
close(3)                                = 0
lstat64("/sys/block/dasda/device", {st_mode=S_IFLNK|0777, st_size=0, ...}) = 0
readlink("/sys/block/dasda/device", "../../devices/css0/0.0.0003/0.0.4e30", 256) = 36
lstat64("/sys/devices/css0/0.0.0003/0.0.4e30", {st_mode=S_IFDIR|0755, st_size=0, ...}) = 0
lstat64("/sys/devices/css0/0.0.0003/0.0.4e30/bus", {st_mode=S_IFLNK|0777, st_size=0, ...}) = 0
readlink("/sys/devices/css0/0.0.0003/0.0.4e30/bus", "../../../../bus/ccw", 256) = 19
lstat64("/sys/devices/css0/0.0.0003/0.0.4e30/driver", {st_mode=S_IFLNK|0777, st_size=0, ...}) = 0
readlink("/sys/devices/css0/0.0.0003/0.0.4e30/driver", "../../../../bus/ccw/drivers/dasd-eckd", 256) =
37
open("/proc/filesystems", O_RDONLY|O_LARGEFILE) = 3
```

```
read(3, "nodev\tsysfs\nnodev\trootfs\nnodev\tb"..., 4095) = 276
close(3)                                = 0
stat64("/dev/dasda", {st_mode=S_IFBLK|0640, st_rdev=makedev(94, 0), ...}) = 0
chmod("/dev/dasda", 060640)             = 0
chown("/dev/dasda", 0, 6)               = 0
...
lstat64("/sys/block/dasda/dasda1", {st_mode=S_IFDIR|0755, st_size=0, ...}) = 0
lstat64("/sys/block/dasda/dasda1/dev", {st_mode=S_IFREG|0444, st_size=4096, ...} ) = 0
stat64("/sys/block/dasda/dasda1/dev", {st_mode=S_IFREG|0444, st_size=4096, ...}) = 0
open("/sys/block/dasda/dasda1/dev", O_RDONLY) = 3
read(3, "94:1\n", 4096)                 = 5
close(3)                                = 0
lstat64("/sys/block/dasda", {st_mode=S_IFDIR|0755, st_size=0, ...}) = 0
lstat64("/sys/block/dasda/device", {st_mode=S_IFLNK|0777, st_size=0, ...}) = 0
readlink("/sys/block/dasda/device", "../../devices/css0/0.0.0003/0.0.4e30", 256) = 36
lstat64("/sys/devices/css0/0.0.0003/0.0.4e30", {st_mode=S_IFDIR|0755, st_size=0, ...}) = 0
lstat64("/sys/devices/css0/0.0.0003/0.0.4e30/bus", {st_mode=S_IFLNK|0777, st_size=0, ...}) = 0
readlink("/sys/devices/css0/0.0.0003/0.0.4e30/bus", "../../../../bus/ccw", 256) = 19
lstat64("/sys/devices/css0/0.0.0003/0.0.4e30/driver", {st_mode=S_IFLNK|0777, st_ size=0, ...}) =
0
readlink("/sys/devices/css0/0.0.0003/0.0.4e30/driver", "../../../../bus/ccw/drivers/dasd-eckd", 256) =
37
stat64("/dev/dasda1", {st_mode=S_IFBLK|0640, st_rdev=makedev(94, 1), ...}) = 0
chmod("/dev/dasda1", 060640)            = 0
chown("/dev/dasda1", 0, 6)              = 0
```

Ideas welcome...

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by Tejun Heo on Wed, 08 Aug 2007 14:35:36 GMT

View Forum Message <> Reply to Message

Does the attached patch happen to fix the problem?

--
tejun

```
 fs/sysfs/dir.c |    4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)

Index: work/fs/sysfs/dir.c
===================================================================
```

```
--- work.orig/fs/sysfs/dir.c
+++ work/fs/sysfs/dir.c
@@ -1010,7 +1010,7 @@ static int sysfs_readdir(struct file * f
   if (filldir(dirent, "..", 2, filp->f_pos, ino, DT_DIR) == 0)
    filp->f_pos++;
  }
- if ((filp->f_pos > 1) && (filp->f_pos < UINT_MAX)) {
+ if ((filp->f_pos > 1) && (filp->f_pos < INT_MAX)) {
   mutex_lock(&sysfs_mutex);

   /* Skip the dentries we have already reported */
@@ -1031,7 +1031,7 @@ static int sysfs_readdir(struct file * f
     break;
   }
   if (!pos)
-   filp->f_pos = UINT_MAX;
+   filp->f_pos = INT_MAX;
   mutex_unlock(&sysfs_mutex);
  }
  return 0;
```

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by Cornelia Huck on Wed, 08 Aug 2007 14:50:27 GMT
View Forum Message <> Reply to Message

On Wed, 08 Aug 2007 23:35:36 +0900,
Tejun Heo <htejun@gmail.com> wrote:

> Does the attached patch happen to fix the problem?

Indeed it does; thanks!

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by Tejun Heo on Wed, 08 Aug 2007 14:55:29 GMT
View Forum Message <> Reply to Message

Cornelia Huck wrote:
> On Wed, 08 Aug 2007 23:35:36 +0900,
> Tejun Heo <htejun@gmail.com> wrote:
>
>> Does the attached patch happen to fix the problem?
>
> Indeed it does; thanks!

Yeah, you seem to have 32bit off_t. UINT_MAX overflows, so...

--
tejun

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

---

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by ebiederm on Wed, 08 Aug 2007 15:08:38 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> Cornelia Huck wrote:
>> On Wed, 08 Aug 2007 23:35:36 +0900,
>> Tejun Heo <htejun@gmail.com> wrote:
>>
>>> Does the attached patch happen to fix the problem?
>>
>> Indeed it does; thanks!
>
> Yeah, you seem to have 32bit off_t. UINT_MAX overflows, so...

Weird. And we have it opening the directory O_LARGEFILE.

I have no problems with the fix though.

Eric

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

---

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by Tejun Heo on Wed, 08 Aug 2007 15:13:37 GMT

Eric W. Biederman wrote:
> Tejun Heo <htejun@gmail.com> writes:
>
>> Cornelia Huck wrote:
>>> On Wed, 08 Aug 2007 23:35:36 +0900,
>>> Tejun Heo <htejun@gmail.com> wrote:
>>>
>>>> Does the attached patch happen to fix the problem?
>>> Indeed it does; thanks!
>> Yeah, you seem to have 32bit off_t. UINT_MAX overflows, so...
>
> Weird. And we have it opening the directory O_LARGEFILE.
>
> I have no problems with the fix though.

It's probably because of struct dirent definition used by readdir(). It may differ depending on archs and glibc version but IIRC the backend implementation of all directory listing functions in glibc were the same. It opens with O_LARGEFILE and use getdents64 to get full info then clip it to whatever limit the called API imposes.

--
tejun

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by Cornelia Huck on Wed, 08 Aug 2007 15:15:00 GMT

On Wed, 08 Aug 2007 23:55:29 +0900,
Tejun Heo <htejun@gmail.com> wrote:

> Yeah, you seem to have 32bit off_t. UINT_MAX overflows, so...

<checks> Yes, that was an 31 bit guest. (Good thing I tried it there, I usually use 64 bit guests...)

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by Tejun Heo on Wed, 08 Aug 2007 15:16:46 GMT
View Forum Message <> Reply to Message

Tejun Heo wrote:
> Eric W. Biederman wrote:
>> Tejun Heo <htejun@gmail.com> writes:
>>
>>> Cornelia Huck wrote:
>>>> On Wed, 08 Aug 2007 23:35:36 +0900,
>>>> Tejun Heo <htejun@gmail.com> wrote:
>>>>
>>>>> Does the attached patch happen to fix the problem?
>>>> Indeed it does; thanks!
>>> Yeah, you seem to have 32bit off_t. UINT_MAX overflows, so...
>> Weird. And we have it opening the directory O_LARGEFILE.
>>
>> I have no problems with the fix though.
>
> It's probably because of struct dirent definition used by readdir().

More specifically, d_off field. It's a bit twisted. For the last
entry, filp->f_pos gets written into the field and gets wrapped while
being copied out to userland or in glibc.

--
tejun

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

Subject: Re: [PATCH 14/25] sysfs: Don't use lookup_one_len_kern
Posted by ebiederm on Wed, 08 Aug 2007 15:26:48 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> On Tue, Aug 07, 2007 at 03:23:57PM -0600, Eric W. Biederman wrote:
>>
>> Upon inspection it appears that there is no looking of the
>> inode mutex in lookup_one_len_kern and we aren't calling
>> it with the inode mutex and that is wrong.
>>
>> So this patch rolls our own dcache insertion function that
>> does exactly what we need it to do. As it turns out this
>> is pretty trivial to do and it makes the code easier to

```
>> audit.
>>
>> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
>> ---
>>  fs/sysfs/dir.c |   41 ++++++++++++++++++++++++++++++++++++++++--
>>  1 files changed, 39 insertions(+), 2 deletions(-)
>>
>> diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
>> index a9bdb12..1d53c2a 100644
>> --- a/fs/sysfs/dir.c
>> +++ b/fs/sysfs/dir.c
>> @@ -765,6 +765,44 @@ static struct dentry *__sysfs_get_dentry(struct
> super_block *sb, struct sysfs_di
>>   return dentry;
>>  }
>>
>> +static struct dentry *sysfs_add_dentry(struct dentry *parent, struct
> sysfs_dirent *sd)
>> +{
>> + struct qstr name;
>> + struct dentry *dentry;
>> + struct inode *inode;
>> +
>> + mutex_lock(&parent->d_inode->i_mutex);
>> + mutex_lock(&sysfs_mutex);
>> + dentry = ERR_PTR(-EINVAL);
>> + if (parent->d_fsdata != sd->s_parent)
>> +  goto out;
>> +
>> + name.name = sd->s_name;
>> + name.len = strlen(sd->s_name);
>> + dentry = d_hash_and_lookup(parent, &name);
>> + if (dentry)
>> +  goto out;
>> +
>> + dentry = d_alloc(parent, &name);
>> + if (!dentry) {
>> +  dentry = ERR_PTR(-ENOMEM);
>> +  goto out;
>> + }
>> +
>> + inode = sysfs_get_inode(sd);
>> + if (!inode) {
>> +  dput(dentry);
>> +  dentry = ERR_PTR(-ENOMEM);
>> +  goto out;
>> + }
>> + d_instantiate(dentry, inode);
```

>> + sysfs_attach_dentry(sd, dentry);
>> +out:
>> + mutex_unlock(&sysfs_mutex);
>> + mutex_unlock(&parent->d_inode->i_mutex);
>> + return dentry;
>> +}
>
> This is virtually identical to
>
> mutex_lock(&parent_dentry->d_inode->i_mutex);
> dentry = lookup_one_len_kern(cur->s_name, parent_dentry,
> strlen(cur->s_name));
> mutex_unlock(&parent_dentry->d_inode->i_mutex);
>
> right? I don't think we need to duplicate the code here. Or is it
> needed for later multi-sb thing?

Right. We can do that as well. In practice in working code
there is no real difference.

There is a little extra uniformity in rolling it ourselves, but
not enough to worry about either way.

In the review/debug etc cycle it just wound up being a lot easier
to roll the code myself.

By the time we get to lookup_one_len_kern it is almost impenetrable
code in namei.c where sysfs_add_dentry tends is easier to comprehend,
and to modify for debugging.

Eric

Subject: Re: [PATCH 17/25] sysfs: Rewrite rename in terms of sysfs dirents
Posted by ebiederm on Wed, 08 Aug 2007 15:32:20 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
> {
> - struct sysfs_dirent *sd;
> + struct sysfs_dirent *sd = kobj->sd;

>>   struct dentry *parent = NULL;
>>   struct dentry *old_dentry = NULL, *new_dentry = NULL;
>>   const char *dup_name = NULL;
>> @@ -863,42 +863,41 @@ int sysfs_rename_dir(struct kobject * kobj, const char
> *new_name)
>>
>>   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(sd);
>>   if (IS_ERR(old_dentry)) {
>>   error = PTR_ERR(old_dentry);
>> + goto out;
>>   }
>>
>>   parent = old_dentry->d_parent;
>>
>>   /* lock parent and get dentry for new name */
>>   mutex_lock(&parent->d_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);
>>   if (!new_name)
>> + goto out_unlock;
>>
>>   error = kobject_set_name(kobj, "%s", new_name);
>>   if (error)
>> + goto out_unlock;
>>
>>   mutex_lock(&sysfs_mutex);
>
> sysfs_mutex is being grabbed twice and unlocked twice later.

Ugh. That patch was supposed to remove the inner grab and release of
sysfs_mutex. I guess I somehow failed to move that change all of the
way down to this patch. #18 gets it right for sysfs_move_dir. And
#20 fixes it but yes this patch is broken, and will mess up any
git-bisect badly.

So this patch needs to be respun.

Eric

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

---

Subject: Re: [PATCH 19/25] sysfs: sysfs_get_dentry add a sb parameter
Posted by ebiederm on Wed, 08 Aug 2007 15:34:27 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> On Tue, Aug 07, 2007 at 03:29:23PM -0600, Eric W. Biederman wrote:
>>
>> In preparation for multiple mounts of sysfs add a superblock parameter to
>> sysfs_get_dentry.
>>
>> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
>> @@ -827,7 +829,7 @@ struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd)
>>   */
>>   dentry = NULL;
>>   cur = sd;
>> - while (!(dentry = __sysfs_get_dentry(sysfs_sb, cur))) {
>> + while (!(dentry = __sysfs_get_dentry(sb, cur))) {
>
> It's probably better to add @sb to __sysfs_get_dentry() here too.
> That will make things look clearer.

A little.

I just wanted to be certain I was testing the final __sysfs_get_dentry
logic as soon as I could.

Eric

_____
Containers mailing list

Subject: Re: [PATCH 14/25] sysfs: Don't use lookup_one_len_kern
Posted by Tejun Heo on Wed, 08 Aug 2007 15:35:39 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
>> right? I don't think we need to duplicate the code here. Or is it
>> needed for later multi-sb thing?
>
> Right. We can do that as well. In practice in working code
> there is no real difference.
>
> There is a little extra uniformity in rolling it ourselves, but
> not enough to worry about either way.
>
> In the review/debug etc cycle it just wound up being a lot easier
> to roll the code myself.
>
> By the time we get to lookup_one_len_kern it is almost impenetrable
> code in namei.c where sysfs_add_dentry tends is easier to comprehend,
> and to modify for debugging.

Yeap, agreed. I agreed with this one too in the comment for the next
patch. I guess I should have wrote here too. Sorry about the trouble.

Thanks.

--
tejun

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

Subject: Re: [PATCH 20/25] sysfs: Rename Support multiple superblocks
Posted by ebiederm on Wed, 08 Aug 2007 15:45:57 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

>> +{
>> + 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)
>> +   continue;
>
> sysfs_get_dentry() return ERR_PTR() value. Oops, sysfs_get_dentry()
> implementation is wrong too. Also, please move
> sysfs_grab/release_supers() near this patch and add (a lot of)
> comments there.
>
> Other than that, I think this is as clean as this can be. Great.

Welcome. I will see what I can do with respect to cleaning up
the names.

As for the return value of sysfs_get_dentry that is tricky. In particular
I have three specific cases the code needs to deal with.

- We got the dentry.
- We did not get the dentry because for this super block there never
  ever will be a dentry.
- Some kind of error occurred in attempting to get the dentry.

Not getting a dentry because it is impossible I am currently handling
with a NULL return. I can equally use a specific error code to mean
that as well. It doesn't much matter. So I guess the hunk in
question could read:

>> + list_for_each_entry(sb, &sysfs_fs_type.fs_supers, s_instances) {
>> + dentry = sysfs_get_dentry(sb, sd);
>> + if (dentry == ERR_PTR(-ENOENT))
>> +   continue;

As long as we handle that class of error differently I really don't
care.

Eric

Subject: Re: [PATCH 20/25] sysfs: Rename Support multiple superblocks

Posted by Tejun Heo on Wed, 08 Aug 2007 15:50:56 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Welcome. I will see what I can do with respect to cleaning up
> the names.
>
> As for the return value of sysfs_get_dentry that is tricky. In particular
> I have three specific cases the code needs to deal with.
>
> - We got the dentry.
> - We did not get the dentry because for this super block there never
>   ever will be a dentry.
> - Some kind of error occurred in attempting to get the dentry.
>
> Not getting a dentry because it is impossible I am currently handling
> with a NULL return. I can equally use a specific error code to mean
> that as well. It doesn't much matter. So I guess the hunk in
> question could read:
>
>>> + list_for_each_entry(sb, &sysfs_fs_type.fs_supers, s_instances) {
>>> + dentry = sysfs_get_dentry(sb, sd);
>>> + if (dentry == ERR_PTR(-ENOENT))
>>> +   continue;
>
> As long as we handle that class of error differently I really don't
> care.

Yeah, I think using -ENOENT is better; otherwise, my little head feels like exploding. :-) More importantly, sysfs_get_dentry() seems like it would deference ERR_PTR() value. No?

--
tejun

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

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by ebiederm on Wed, 08 Aug 2007 15:53:38 GMT
View Forum Message <> Reply to Message

Tejun Heo <teheo@suse.de> writes:

> Tejun Heo wrote:
>> Eric W. Biederman wrote:

>>> Tejun Heo <htejun@gmail.com> writes:
>>>
>>>> Cornelia Huck wrote:
>>>>> On Wed, 08 Aug 2007 23:35:36 +0900,
>>>>> Tejun Heo <htejun@gmail.com> wrote:
>>>>>
>>>>>> Does the attached patch happen to fix the problem?
>>>>> Indeed it does; thanks!
>>>> Yeah, you seem to have 32bit off_t. UINT_MAX overflows, so...
>>> Weird. And we have it opening the directory O_LARGEFILE.
>>>
>>> I have no problems with the fix though.
>>
>> It's probably because of struct dirent definition used by readdir().
>
> More specifically, d_off field. It's a bit twisted. For the last
> entry, filp->f_pos gets written into the field and gets wrapped while
> being copied out to userland or in glibc.

That could do it, and glibc is crunching it. Oh well, it is
easy enough to avoid as long as our inode numbers are small which
the idr allocator seems to ensure.

Eric

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

---

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by Tejun Heo on Wed, 08 Aug 2007 16:03:20 GMT
View Forum Message <> Reply to Message

Hello,

Eric W. Biederman wrote:
>> More specifically, d_off field. It's a bit twisted. For the last
>> entry, filp->f_pos gets written into the field and gets wrapped while
>> being copied out to userland or in glibc.
>
> That could do it, and glibc is crunching it. Oh well, it is
> easy enough to avoid as long as our inode numbers are small which
> the idr allocator seems to ensure.

Yeah, now I think about it. glibc throws out entries which don't fit in
the data structure specified by the called API, so it probably threw out
the last entry which has UINT_MAX in d_off which doesn't fit in the

readdir() return structure. Using INT_MAX should be just fine as IDA always allocates the first empty slot. We can add paranoia check in ino allocation path.

--
tejun

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

---

Subject: Re: alternative approached at tagged nodes
Posted by Tejun Heo on Wed, 08 Aug 2007 16:31:53 GMT
View Forum Message <> Reply to Message

Okay, here's a different implementation of tagged nodes. I just compile tested it so I can guarantee it's broken. I don't think there's anything fundamentally wrong but I'm wrong fairly often, so if you find something moronic, feel free to scream at me.

1. there's no enable_tagging() or fundamental restrictions on which types of nodes can be tagged.

2. no callback. tags are set by sysfs_rename_dir_tagged(). symlinks follow the tag of its target_sd. This limits taggable node types to dirs and symlinks.

3. in all paths between the root and leaf nodes, only zero or one tagged sd exists. all children of a tagged sd have NULL s_tag but are considered tagged the same as the tagged ancestor.

4. untagged entries are visible in all supers unless there's a matching tagged entry overshadowing it. tagged entry is only visible in the matching tagged super.

5. due to #3, children in a directory which belong to the same tag can be looked up using NULL tag. Most leaf node ops don't need to be changed.

6. symlink removal is different. we either need to modify the interface to take target_sd or implement new one. actually, I think linking symlinks to target_sd and renaming/removing them automagically would be pretty good.

Thanks. What do you think?

JUST FOR BRAIN STORMING. DO NOT APPLY.

```
---
 fs/sysfs/bin.c        |    2
 fs/sysfs/dir.c        |  122 ++++++++++++++++++++++++++++++++++++++++++++----------
 fs/sysfs/file.c       |    4 -
 fs/sysfs/group.c      |    2
 fs/sysfs/inode.c      |    5 +-
 fs/sysfs/mount.c      |   44 ++++++++++++++++++--
 fs/sysfs/symlink.c    |    8 ++-
 fs/sysfs/sysfs.h      |   11 +++-
 include/linux/sysfs.h |    4 +
 9 files changed, 167 insertions(+), 35 deletions(-)

Index: work/fs/sysfs/dir.c
===================================================================
--- work.orig/fs/sysfs/dir.c
+++ work/fs/sysfs/dir.c
@@ -387,7 +387,7 @@ 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))
+ if (__sysfs_find_dirent(acxt->parent_sd, sd->s_name, sd->s_tag, 0))
   return -EEXIST;

  sd->s_parent = sysfs_get(acxt->parent_sd);
@@ -526,6 +526,24 @@ void sysfs_addrm_finish(struct sysfs_add
  }
 }

+struct sysfs_dirent *__sysfs_find_dirent(struct sysfs_dirent *parent_sd,
+      const unsigned char *name,
+      const void *tag, int match_null)
+{
+ struct sysfs_dirent *null_sd = NULL, *sd;
+
+ for (sd = parent_sd->s_children; sd; sd = sd->s_sibling) {
+  if (strcmp(sd->s_name, name))
+   continue;
+  if (sd->s_tag == tag)
+   return sd;
+  if (!sd->s_tag && match_null)
+   null_sd = sd;
+ }
+
+ return null_sd;
+}
+
 /**
```

```
  * sysfs_find_dirent - find sysfs_dirent with the given name
  * @parent_sd: sysfs_dirent to search under
@@ -542,12 +560,7 @@ void sysfs_addrm_finish(struct sysfs_add
 struct sysfs_dirent *sysfs_find_dirent(struct sysfs_dirent *parent_sd,
            const unsigned char *name)
 {
- struct sysfs_dirent *sd;
-
- for (sd = parent_sd->s_children; sd; sd = sd->s_sibling)
-  if (!strcmp(sd->s_name, name))
-   return sd;
- return NULL;
+ return __sysfs_find_dirent(parent_sd, name, NULL, 0);
 }

 /**
@@ -642,7 +655,8 @@ static struct dentry * sysfs_lookup(stru

  mutex_lock(&sysfs_mutex);

- sd = sysfs_find_dirent(parent_sd, dentry->d_name.name);
+ sd = __sysfs_find_dirent(parent_sd, dentry->d_name.name,
+      sysfs_info(dentry->d_sb)->tag, 1);

  /* no such entry */
  if (!sd)
@@ -803,9 +817,19 @@ out:
  return dentry;
 }

+const void *sysfs_dirent_tag(struct sysfs_dirent *sd)
+{
+ while (sd) {
+  if (sd->s_tag)
+   return sd->s_tag;
+  sd = sd->s_parent;
+ }
+ return sd->s_tag;
+}
+
 /**
  * sysfs_get_dentry - get dentry for the given sysfs_dirent
- * @sb: superblock of the dentry to return
+ * @super: superblock of the dentry to return
  * @sd: sysfs_dirent of interest
  *
  * Get dentry for @sd.  Dentry is looked up if currently not
@@ -820,16 +844,23 @@ out:
```

```
  * Pointer to found dentry on success, ERR_PTR() value on error.
  * NULL if the sysfs dentry does not appear in the specified superblock
  */
-struct dentry *sysfs_get_dentry(struct super_block *sb, struct sysfs_dirent *sd)
+struct dentry *sysfs_get_dentry(struct super_block *super,
+    struct sysfs_dirent *sd)
 {
+ const void *tag;
  struct sysfs_dirent *cur;
  struct dentry *parent_dentry, *dentry;

+ /* bail if this sd won't show up in this superblock */
+ tag = sysfs_dirent_tag(sd);
+ if (tag && tag != sysfs_info(super)->tag)
+  return ERR_PTR(-ENOENT);
+
  /* Find the first parent which has valid dentry.
   */
  dentry = NULL;
  cur = sd;
- while (!(dentry = __sysfs_get_dentry(sb, cur))) {
+ while (!(dentry = __sysfs_get_dentry(super, cur))) {
   if (cur->s_flags & SYSFS_FLAG_REMOVED) {
    dentry = ERR_PTR(-ENOENT);
    break;
@@ -926,27 +957,44 @@ err_out:
  return error;
 }

-
-int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
+int sysfs_rename_dir_tagged(struct kobject * kobj, const char *new_name,
+       const void *new_tag)
 {
  struct sysfs_dirent *sd = kobj->sd;
  struct list_head todo;
  struct sysfs_rename_struct *srs;
  struct inode *parent_inode = NULL;
  const char *dup_name = NULL;
- int error;
+ const void *old_tag;
+ int tag = 0, error;

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

+ old_tag = sysfs_dirent_tag(sd);
+
```

```
+ /* need tagging? */
+ if (old_tag != new_tag) {
+  tag = 1;
+
+  /* can't tag again in tagged subtree */
+  error = -EINVAL;
+  if (old_tag != new_tag)
+   goto out;
+ } else
+  /* sd->s_tag can be either old_tag or NULL */
+  new_tag = sd->s_tag;
+
  error = 0;
- if (strcmp(sd->s_name, new_name) == 0)
+ if (!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 (!tag) {
+  error = prep_rename(&todo, sd, sd->s_parent, new_name);
+  if (error)
+   goto out_release;
+ }

  error = -ENOMEM;
  mutex_lock(&sysfs_mutex);
@@ -959,7 +1007,7 @@ 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, new_name, new_tag, 0))
   goto out_unlock;

  /* rename kobject and sysfs_dirent */
@@ -974,6 +1022,7 @@ int sysfs_rename_dir(struct kobject * ko

  dup_name = sd->s_name;
  sd->s_name = new_name;
+ sd->s_tag = new_tag;

  /* rename */
  list_for_each_entry(srs, &todo, list) {
@@ -981,6 +1030,21 @@ 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 (tag) {
+  struct super_block *super;
+  struct dentry *dentry;
+
+  list_for_each_entry(super, &sysfs_fs_type.fs_supers, s_instances) {
+   dentry = __sysfs_get_dentry(super, sd);
+   if (!dentry)
+    continue;
+   shrink_dcache_parent(dentry);
+   d_drop(dentry);
+   dput(dentry);
+  }
+ }
+
  error = 0;
 out_unlock:
  mutex_unlock(&sysfs_mutex);
@@ -995,6 +1059,11 @@ out:
  return error;
 }

+int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
+{
+ return sysfs_rename_dir_tagged(kobj, new_name, kobj->sd->s_tag);
+}
+
 int sysfs_move_dir(struct kobject *kobj, struct kobject *new_parent_kobj)
 {
  struct sysfs_dirent *sd = kobj->sd;
@@ -1013,6 +1082,11 @@ int sysfs_move_dir(struct kobject *kobj,
  if (sd->s_parent == new_parent_sd)
   goto out; /* nothing to move */

+ /* can't move between parents belonging to different tags */
+ error = -EINVAL;
+ if (sysfs_dirent_tag(sd->s_parent) != sysfs_dirent_tag(new_parent_sd))
+  goto out;
+
  sysfs_grab_supers();
  error = prep_rename(&todo, sd, new_parent_sd, sd->s_name);
  if (error)
@@ -1041,7 +1115,7 @@ again:
  mutex_lock(&sysfs_mutex);

  error = -EEXIST;
```

```
- if (sysfs_find_dirent(new_parent_sd, sd->s_name))
+ if (__sysfs_find_dirent(new_parent_sd, sd->s_name, sd->s_tag, 0))
   goto out_unlock;

  error = 0;
@@ -1080,8 +1154,9 @@ 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;
+ const void *tag = sysfs_info(filp->f_dentry->d_sb)->tag;
  struct sysfs_dirent *pos;
  ino_t ino;

@@ -1110,6 +1185,9 @@ static int sysfs_readdir(struct file * f
    const char * name;
    int len;

+   if (pos->s_tag && pos->s_tag != tag)
+    continue;
+
    name = pos->s_name;
    len = strlen(name);
    filp->f_pos = ino = pos->s_ino;
Index: work/fs/sysfs/sysfs.h
===================================================================
--- work.orig/fs/sysfs/sysfs.h
+++ work/fs/sysfs/sysfs.h
@@ -26,6 +26,7 @@ struct sysfs_dirent {
  struct sysfs_dirent * s_sibling;
  struct sysfs_dirent * s_children;
  const char  * s_name;
+ const void  * s_tag;

  union {
   struct sysfs_elem_dir  dir;
@@ -51,7 +52,8 @@ struct sysfs_addrm_cxt {
 };

 struct sysfs_super_info {
- int grabbed;
+ int  grabbed;
+ const void *tag;
 };
```

```
 #define sysfs_info(SB) ((struct sysfs_super_info *)(SB)->s_fs_info)
@@ -63,6 +65,7 @@ extern struct file_system_type sysfs_fs_
 void sysfs_grab_supers(void);
 void sysfs_release_supers(void);

+extern const void *sysfs_dirent_tag(struct sysfs_dirent *sd);
 extern struct dentry *sysfs_get_dentry(struct super_block *sb, struct sysfs_dirent *sd);
 extern struct sysfs_dirent *sysfs_get_active(struct sysfs_dirent *sd);
 extern void sysfs_put_active(struct sysfs_dirent *sd);
@@ -79,6 +82,9 @@ extern void sysfs_addrm_finish(struct sy
 extern struct inode * sysfs_get_inode(struct sysfs_dirent *sd);

 extern void release_sysfs_dirent(struct sysfs_dirent * sd);
+extern struct sysfs_dirent *__sysfs_find_dirent(struct sysfs_dirent *parent_sd,
+      const unsigned char *name,
+      const void *tag, int match_null);
 extern struct sysfs_dirent *sysfs_find_dirent(struct sysfs_dirent *parent_sd,
           const unsigned char *name);
 extern struct sysfs_dirent *sysfs_get_dirent(struct sysfs_dirent *parent_sd,
@@ -88,7 +94,8 @@ extern struct sysfs_dirent *sysfs_new_di

 extern int sysfs_add_file(struct sysfs_dirent *dir_sd,
      const struct attribute *attr, int type);
-extern int sysfs_hash_and_remove(struct sysfs_dirent *dir_sd, const char *name);
+extern int sysfs_hash_and_remove(struct sysfs_dirent *dir_sd, const char *name,
+     const void *tag);
 extern struct sysfs_dirent *sysfs_find(struct sysfs_dirent *dir, const char * name);

 extern int sysfs_create_subdir(struct kobject *kobj, const char *name,
Index: work/fs/sysfs/bin.c
===================================================================
--- work.orig/fs/sysfs/bin.c
+++ work/fs/sysfs/bin.c
@@ -248,7 +248,7 @@ int sysfs_create_bin_file(struct kobject

 void sysfs_remove_bin_file(struct kobject * kobj, struct bin_attribute * attr)
 {
- if (sysfs_hash_and_remove(kobj->sd, attr->attr.name) < 0) {
+ if (sysfs_hash_and_remove(kobj->sd, attr->attr.name, NULL) < 0) {
   printk(KERN_ERR "%s: "
    "bad dentry or inode or no such file: \"%s\"\n",
    __FUNCTION__, attr->attr.name);
Index: work/fs/sysfs/file.c
===================================================================
--- work.orig/fs/sysfs/file.c
+++ work/fs/sysfs/file.c
@@ -560,7 +560,7 @@ 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->sd, attr->name, NULL);
 }


@@ -577,7 +577,7 @@ void sysfs_remove_file_from_group(struct

  dir_sd = sysfs_get_dirent(kobj->sd, group);
  if (dir_sd) {
-  sysfs_hash_and_remove(dir_sd, attr->name);
+  sysfs_hash_and_remove(dir_sd, attr->name, NULL);
   sysfs_put(dir_sd);
  }
 }
Index: work/fs/sysfs/group.c
===================================================================
--- work.orig/fs/sysfs/group.c
+++ work/fs/sysfs/group.c
@@ -23,7 +23,7 @@ static void remove_files(struct sysfs_di
  struct attribute *const* attr;

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

 static int create_files(struct sysfs_dirent *dir_sd,
Index: work/fs/sysfs/inode.c
===================================================================
--- work.orig/fs/sysfs/inode.c
+++ work/fs/sysfs/inode.c
@@ -212,7 +212,8 @@ 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 sysfs_dirent *dir_sd, const char *name,
+      const void *tag)
 {
  struct sysfs_addrm_cxt acxt;
  struct sysfs_dirent *sd;
@@ -222,7 +223,7 @@ int sysfs_hash_and_remove(struct sysfs_d

  sysfs_addrm_start(&acxt, dir_sd);

- sd = sysfs_find_dirent(dir_sd, name);
```

```
+ sd = __sysfs_find_dirent(dir_sd, name, tag, 0);
  if (sd)
   sysfs_remove_one(&acxt, sd);

Index: work/fs/sysfs/mount.c
===================================================================
--- work.orig/fs/sysfs/mount.c
+++ work/fs/sysfs/mount.c
@@ -67,6 +67,7 @@ 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;
@@ -80,20 +81,55 @@ 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: work/fs/sysfs/symlink.c
===================================================================
--- work.orig/fs/sysfs/symlink.c
+++ work/fs/sysfs/symlink.c
@@ -87,6 +87,7 @@ int sysfs_create_link(struct kobject * k
   goto out_put;

  sd->s_elem.symlink.target_sd = target_sd;
+ sd->s_tag = sysfs_dirent_tag(target_sd);
  target_sd = NULL; /* reference is now owned by the symlink */

  sysfs_addrm_start(&acxt, parent_sd);
@@ -113,9 +114,14 @@ int sysfs_create_link(struct kobject * k

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

```
+/**
+ * XXX - we need
+ * sysfs_delete_link(struct kobject *kobj, struct kobject *target);
+ */
+
 static int sysfs_get_target_path(struct sysfs_dirent * parent_sd,
      struct sysfs_dirent * target_sd, char *path)
 {
Index: work/include/linux/sysfs.h
===================================================================
--- work.orig/include/linux/sysfs.h
+++ work/include/linux/sysfs.h
@@ -97,6 +97,10 @@ extern void
 sysfs_remove_dir(struct kobject *);

 extern int __must_check
+sysfs_rename_dir_tagged(struct kobject *kobj, const char *new_name,
+   const void *new_tag);
+
+extern int __must_check
 sysfs_rename_dir(struct kobject *kobj, const char *new_name);

 extern int __must_check
```

Subject: Re: [PATCH 20/25] sysfs: Rename Support multiple superblocks
Posted by ebiederm on Wed, 08 Aug 2007 16:35:38 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> Yeah, I think using -ENOENT is better; otherwise, my little head feels
> like exploding. :-)

Ok. I think I know the feeling.

> More importantly, sysfs_get_dentry() seems like it
> would deference ERR_PTR() value. No?

I'm confused where you are referring to but I will try answer
this.

__sysfs_get_dentry always returns a dentry or NULL if it can't
find the dentry.

I have quoted my final version of sysfs_get_dentry below for discussion.

> struct dentry *sysfs_get_dentry(struct super_block *sb, struct sysfs_dirent *sd)
> {
> struct sysfs_dirent *cur;
> struct dentry *parent_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 NULL;
> }

This is the only location where could return NULL the early check
if this operation is possible.

> /* Find the first parent which has valid dentry.
>   */
> dentry = NULL;
> cur = sd;
> while (!(dentry = __sysfs_get_dentry(sb, cur))) {
>   if (cur->s_flags & SYSFS_FLAG_REMOVED) {
>   dentry = ERR_PTR(-ENOENT);
>   break;
>   }
>   cur = cur->s_parent;
> }

Here we depend on the fact that sysfs_root is pointed to
by sb->s_root so we know it will always have a dentry.

> /* from the found dentry, look up depth times */
> while (dentry->d_fsdata != sd) {
>   /* Find the first ancestor I have not looked up */
>   cur = sd;
>   while (cur->s_parent != dentry->d_fsdata)
>   cur = cur->s_parent;
>
>   /* look it up */
>   parent_dentry = dentry;
>   dentry = sysfs_add_dentry(parent_dentry, cur);
>   dput(parent_dentry);
>
>   if (IS_ERR(dentry))

>   break;
> }
> return dentry;
> }

Eric

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

---

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by Cornelia Huck on Wed, 08 Aug 2007 16:35:43 GMT
View Forum Message <> Reply to Message

On Wed, 8 Aug 2007 16:50:27 +0200,
Cornelia Huck <cornelia.huck@de.ibm.com> wrote:

> On Wed, 08 Aug 2007 23:35:36 +0900,
> Tejun Heo <htejun@gmail.com> wrote:
>
> > Does the attached patch happen to fix the problem?
>
> Indeed it does; thanks!

And the whole patchset seems to work as well. I did my usual
attach/detach/move stuff, and didn't see any problems.

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

---

Subject: Re: [PATCH 0/25] Sysfs cleanups & tagged directory support
Posted by ebiederm on Wed, 08 Aug 2007 16:37:10 GMT
View Forum Message <> Reply to Message

Tejun Heo <teheo@suse.de> writes:

> Hello,
>
> Eric W. Biederman wrote:
>>> More specifically, d_off field. It's a bit twisted. For the last
>>> entry, filp->f_pos gets written into the field and gets wrapped while
>>> being copied out to userland or in glibc.
>>

>> That could do it, and glibc is crunching it. Oh well, it is
>> easy enough to avoid as long as our inode numbers are small which
>> the idr allocator seems to ensure.
>
> Yeah, now I think about it. glibc throws out entries which don't fit in
> the data structure specified by the called API, so it probably threw out
> the last entry which has UINT_MAX in d_off which doesn't fit in the
> readdir() return structure. Using INT_MAX should be just fine as IDA
> always allocates the first empty slot. We can add paranoia check in ino
> allocation path.

Sounds reasonable. Annoying but reasonable.

Eric

Subject: Re: [PATCH 20/25] sysfs: Rename Support multiple superblocks
Posted by Tejun Heo on Wed, 08 Aug 2007 16:42:03 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
>> /* Find the first parent which has valid dentry.
>>   */
>> dentry = NULL;
>> cur = sd;
>> while (!(dentry = __sysfs_get_dentry(sb, cur))) {
>>   if (cur->s_flags & SYSFS_FLAG_REMOVED) {
>>   dentry = ERR_PTR(-ENOENT);
>>   break;
>>   }
>>   cur = cur->s_parent;
>> }
>
> Here we depend on the fact that sysfs_root is pointed to
> by sb->s_root so we know it will always have a dentry.

Hmmm... dentry could be ERR_PTR(-ENOENT) here if the REMOVED flag test
succeeded, right?

>> /* from the found dentry, look up depth times */
>> while (dentry->d_fsdata != sd) {

And then dereferenced. The REMOVED test should never succeed there, so
we're probably in the clear but still the code looks a bit scary.

--
tejun

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

---

Subject: Re: [PATCH 20/25] sysfs: Rename Support multiple superblocks
Posted by ebiederm on Wed, 08 Aug 2007 16:55:10 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> Eric W. Biederman wrote:
>>> /* Find the first parent which has valid dentry.
>>>   */
>>> dentry = NULL;
>>> cur = sd;
>>> while (!(dentry = __sysfs_get_dentry(sb, cur))) {
>>>   if (cur->s_flags & SYSFS_FLAG_REMOVED) {
>>>   dentry = ERR_PTR(-ENOENT);
>>>   break;
>>>   }
>>>   cur = cur->s_parent;
>>> }
>>
>> Here we depend on the fact that sysfs_root is pointed to
>> by sb->s_root so we know it will always have a dentry.
>
> Hmmm... dentry could be ERR_PTR(-ENOENT) here if the REMOVED flag test
> succeeded, right?

Ugh right. Now that I don't have the locking it probably makes
sense to have that path just return or branch to the exit.

>
>>> /* from the found dentry, look up depth times */
>>> while (dentry->d_fsdata != sd) {
>
> And then dereferenced. The REMOVED test should never succeed there, so
> we're probably in the clear but still the code looks a bit scary.

Agreed. That is a bug. Either the removed test needs to
be removed because it can't happen or that case needs to be handled.

Thanks for spotting this one.

Mental blind spots are the worst.

Eric

_____

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