Subject: [PATCH 0/14] sysfs cleanups
Posted by ebiederm on Tue, 31 Jul 2007 10:16:25 GMT
View Forum Message <> Reply to Message

Ok. I don't know how alert I am at the moment (it's way past my bed time) but I have been digging in and cleaning up sysfs. I'm not quite to the point of supporting multiple mounts and shadow directories again but the locking is looking much simpler and cleaner. So with a little luck this will be a better base to build on.

If something looks horrible please holler.

Eric

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

Subject: [PATCH 01/14] sysfs: Remove first pass at shadow directory support
Posted by ebiederm on Tue, 31 Jul 2007 10:18:35 GMT
View Forum Message <> Reply to Message

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

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

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
I have sanitized my changes to sysfs_rename_dir since last time I posted this patch.

```
 fs/sysfs/dir.c          |  155 ++----------------------------------------
 fs/sysfs/group.c        |    1 -
 fs/sysfs/inode.c        |   10 ---
 fs/sysfs/mount.c        |    2 +-
 fs/sysfs/sysfs.h        |    6 --
 include/linux/kobject.h |    5 --
 include/linux/sysfs.h   |   27 +-------
 lib/kobject.c           |   44 ++-----------
 8 files changed, 18 insertions(+), 232 deletions(-)
```

```
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 048e605..40b9efe 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -569,9 +569,6 @@ static void sysfs_drop_dentry(struct sysfs_dirent *sd)
  spin_unlock(&dcache_lock);
  spin_unlock(&sysfs_assoc_lock);

- /* dentries for shadowed inodes are pinned, unpin */
- if (dentry && sysfs_is_shadowed_inode(dentry->d_inode))
-  dput(dentry);
  dput(dentry);

  /* adjust nlink and update timestamp */
@@ -723,19 +720,15 @@ int sysfs_create_subdir(struct kobject *kobj, const char *name,
 /**
  * sysfs_create_dir - create a directory for an object.
  * @kobj:  object we're creating directory for.
- * @shadow_parent: parent object.
  */
-int sysfs_create_dir(struct kobject *kobj,
-       struct sysfs_dirent *shadow_parent_sd)
+int sysfs_create_dir(struct kobject * kobj)
 {
  struct sysfs_dirent *parent_sd, *sd;
  int error = 0;

  BUG_ON(!kobj);

- if (shadow_parent_sd)
-  parent_sd = shadow_parent_sd;
- else if (kobj->parent)
+ 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;
@@ -887,8 +880,7 @@ void sysfs_remove_dir(struct kobject * kobj)
  __sysfs_remove_dir(sd);
 }

-int sysfs_rename_dir(struct kobject *kobj, struct sysfs_dirent *new_parent_sd,
-       const char *new_name)
+int sysfs_rename_dir(struct kobject *kobj, const char *new_name)
 {
  struct sysfs_dirent *sd = kobj->sd;
  struct dentry *new_parent = NULL;
@@ -903,11 +895,7 @@ int sysfs_rename_dir(struct kobject *kobj, struct sysfs_dirent
```

```
*new_parent_sd,
   goto out_dput;
  }

- new_parent = sysfs_get_dentry(new_parent_sd);
- if (IS_ERR(new_parent)) {
-  error = PTR_ERR(new_parent);
-  goto out_dput;
- }
+ new_parent = dget(old_dentry->d_parent);

  /* lock new_parent and get dentry for new name */
  mutex_lock(&new_parent->d_inode->i_mutex);
@@ -918,14 +906,8 @@ int sysfs_rename_dir(struct kobject *kobj, struct sysfs_dirent
*new_parent_sd,
   goto out_unlock;
  }

- /* By allowing two different directories with the same
-  * d_parent we allow this routine to move between different
-  * shadows of the same directory
-  */
  error = -EINVAL;
- if (old_dentry->d_parent->d_inode != new_parent->d_inode ||
-     new_dentry->d_parent->d_inode != new_parent->d_inode ||
-     old_dentry == new_dentry)
+ if (old_dentry == new_dentry)
   goto out_unlock;

  error = -EEXIST;
@@ -942,23 +924,15 @@ int sysfs_rename_dir(struct kobject *kobj, struct sysfs_dirent
*new_parent_sd,
  if (error)
   goto out_drop;

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

  /* move under the new parent */
  d_add(new_dentry, NULL);
  d_move(sd->s_dentry, new_dentry);

- 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);
-
  error = 0;
  goto out_unlock;

@@ -1189,121 +1163,6 @@ static loff_t sysfs_dir_lseek(struct file * file, loff_t offset, int origin)
  return offset;
 }

-
-/**
- * sysfs_make_shadowed_dir - Setup so a directory can be shadowed
- * @kobj: object we're creating shadow of.
- */
-
-int sysfs_make_shadowed_dir(struct kobject *kobj,
- void * (*follow_link)(struct dentry *, struct nameidata *))
-{
- struct dentry *dentry;
- struct inode *inode;
- struct inode_operations *i_op;
-
- /* get dentry for @kobj->sd, dentry of a shadowed dir is pinned */
- dentry = sysfs_get_dentry(kobj->sd);
- if (IS_ERR(dentry))
-  return PTR_ERR(dentry);
-
- inode = dentry->d_inode;
- if (inode->i_op != &sysfs_dir_inode_operations) {
-  dput(dentry);
-  return -EINVAL;
- }
-
- i_op = kmalloc(sizeof(*i_op), GFP_KERNEL);
- if (!i_op)
-  return -ENOMEM;
-
- memcpy(i_op, &sysfs_dir_inode_operations, sizeof(*i_op));
- i_op->follow_link = follow_link;
-
- /* Locking of inode->i_op?
-  * Since setting i_op is a single word write and they
-  * are atomic we should be ok here.
-  */
```

```
- inode->i_op = i_op;
- return 0;
-}
-
-/**
- * sysfs_create_shadow_dir - create a shadow directory for an object.
- * @kobj: object we're creating directory for.
- *
- * sysfs_make_shadowed_dir must already have been called on this
- * directory.
- */
-
-struct sysfs_dirent *sysfs_create_shadow_dir(struct kobject *kobj)
-{
- struct sysfs_dirent *parent_sd = kobj->sd->s_parent;
- struct dentry *dir, *parent, *shadow;
- struct inode *inode;
- struct sysfs_dirent *sd;
- struct sysfs_addrm_cxt acxt;
-
- dir = sysfs_get_dentry(kobj->sd);
- if (IS_ERR(dir)) {
-  sd = (void *)dir;
-  goto out;
- }
- parent = dir->d_parent;
-
- inode = dir->d_inode;
- sd = ERR_PTR(-EINVAL);
- if (!sysfs_is_shadowed_inode(inode))
-  goto out_dput;
-
- shadow = d_alloc(parent, &dir->d_name);
- if (!shadow)
-  goto nomem;
-
- sd = sysfs_new_dirent("_SHADOW_", inode->i_mode, SYSFS_DIR);
- if (!sd)
-  goto nomem;
- sd->s_elem.dir.kobj = kobj;
-
- sysfs_addrm_start(&acxt, parent_sd);
-
- /* add but don't link into children list */
- sysfs_add_one(&acxt, sd);
-
- /* attach and instantiate dentry */
- sysfs_attach_dentry(sd, shadow);
```

```
- d_instantiate(shadow, igrab(inode));
- inc_nlink(inode); /* tj: synchronization? */
-
- sysfs_addrm_finish(&acxt);
-
- dget(shadow);  /* Extra count - pin the dentry in core */
-
- goto out_dput;
-
- nomem:
- dput(shadow);
- sd = ERR_PTR(-ENOMEM);
- out_dput:
- dput(dir);
- out:
- return sd;
-}
-
-/**
- * sysfs_remove_shadow_dir - remove an object's directory.
- * @shadow_sd: sysfs_dirent of shadow directory
- *
- * The only thing special about this is that we remove any files in
- * the directory before we remove the directory, and we've inlined
- * what used to be sysfs_rmdir() below, instead of calling separately.
- */
-
-void sysfs_remove_shadow_dir(struct sysfs_dirent *shadow_sd)
-{
- __sysfs_remove_dir(shadow_sd);
-}
-
 const struct file_operations sysfs_dir_operations = {
  .open  = sysfs_dir_open,
  .release = sysfs_dir_close,
diff --git a/fs/sysfs/group.c b/fs/sysfs/group.c
index f318b73..4606f7c 100644
--- a/fs/sysfs/group.c
+++ b/fs/sysfs/group.c
@@ -13,7 +13,6 @@
 #include <linux/dcache.h>
 #include <linux/namei.h>
 #include <linux/err.h>
-#include <linux/fs.h>
 #include <asm/semaphore.h>
 #include "sysfs.h"

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

```
index 10d1b52..9671164 100644
--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -34,16 +34,6 @@ static const struct inode_operations sysfs_inode_operations ={
  .setattr = sysfs_setattr,
 };

-void sysfs_delete_inode(struct inode *inode)
-{
- /* Free the shadowed directory inode operations */
- if (sysfs_is_shadowed_inode(inode)) {
-  kfree(inode->i_op);
-  inode->i_op = NULL;
- }
- return generic_delete_inode(inode);
-}
-
 int sysfs_setattr(struct dentry * dentry, struct iattr * iattr)
 {
  struct inode * inode = dentry->d_inode;
diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index fbc7b65..0c016e1 100644
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -21,7 +21,7 @@ struct kmem_cache *sysfs_dir_cachep;

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

 struct sysfs_dirent sysfs_root = {
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index 6b8c8d7..b55e510 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -70,7 +70,6 @@ extern void sysfs_remove_one(struct sysfs_addrm_cxt *acxt,
        struct sysfs_dirent *sd);
 extern int sysfs_addrm_finish(struct sysfs_addrm_cxt *acxt);

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

@@ -121,8 +120,3 @@ static inline void sysfs_put(struct sysfs_dirent * sd)
  if (sd && atomic_dec_and_test(&sd->s_count))
   release_sysfs_dirent(sd);
```

```
 }
-
-static inline int sysfs_is_shadowed_inode(struct inode *inode)
-{
- return S_ISDIR(inode->i_mode) && inode->i_op->follow_link;
-}
diff --git a/include/linux/kobject.h b/include/linux/kobject.h
index aa2fe22..7108a3e 100644
--- a/include/linux/kobject.h
+++ b/include/linux/kobject.h
@@ -80,14 +80,9 @@ extern void kobject_init(struct kobject *);
 extern void kobject_cleanup(struct kobject *);

 extern int __must_check kobject_add(struct kobject *);
-extern int __must_check kobject_shadow_add(struct kobject *kobj,
-       struct sysfs_dirent *shadow_parent);
 extern void kobject_del(struct kobject *);

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

 extern int __must_check kobject_register(struct kobject *);
diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
index be8228e..c16e4c5 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -17,9 +17,6 @@

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

 /* FIXME
  * The *owner field is no longer used, but leave around
@@ -94,14 +91,13 @@ extern int sysfs_schedule_callback(struct kobject *kobj,
   void (*func)(void *), void *data, struct module *owner);

 extern int __must_check
-sysfs_create_dir(struct kobject *kobj, struct sysfs_dirent *shadow_parent_sd);
+sysfs_create_dir(struct kobject *);

 extern void
 sysfs_remove_dir(struct kobject *);
```

```
 extern int __must_check
-sysfs_rename_dir(struct kobject *kobj, struct sysfs_dirent *new_parent_sd,
-   const char *new_name);
+sysfs_rename_dir(struct kobject *kobj, const char *new_name);

 extern int __must_check
 sysfs_move_dir(struct kobject *, struct kobject *);
@@ -138,12 +134,6 @@ void sysfs_remove_file_from_group(struct kobject *kobj,

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

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

 #else /* CONFIG_SYSFS */
@@ -154,8 +144,7 @@ static inline int sysfs_schedule_callback(struct kobject *kobj,
  return -ENOSYS;
 }

-static inline int sysfs_create_dir(struct kobject *kobj,
-       struct sysfs_dirent *shadow_parent_sd)
+static inline int sysfs_create_dir(struct kobject * kobj)
 {
  return 0;
 }
@@ -165,9 +154,7 @@ static inline void sysfs_remove_dir(struct kobject * k)
  ;
 }

-static inline int sysfs_rename_dir(struct kobject *kobj,
-       struct sysfs_dirent *new_parent_sd,
-       const char *new_name)
+static inline int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
 {
  return 0;
 }
@@ -242,12 +229,6 @@ static inline void sysfs_notify(struct kobject * k, char *dir, char *attr)
 {
 }

-static inline int sysfs_make_shadowed_dir(struct kobject *kobj,
- void * (*follow_link)(struct dentry *, struct nameidata *))
```

```
-{
- return 0;
-}
-
 static inline int __must_check sysfs_init(void)
 {
  return 0;
diff --git a/lib/kobject.c b/lib/kobject.c
index 4b08e0f..2fc9fc6 100644
--- a/lib/kobject.c
+++ b/lib/kobject.c
@@ -44,11 +44,11 @@ static int populate_dir(struct kobject * kobj)
  return error;
 }

-static int create_dir(struct kobject *kobj, struct sysfs_dirent *shadow_parent)
+static int create_dir(struct kobject * kobj)
 {
  int error = 0;
  if (kobject_name(kobj)) {
-  error = sysfs_create_dir(kobj, shadow_parent);
+  error = sysfs_create_dir(kobj);
    if (!error) {
     if ((error = populate_dir(kobj)))
      sysfs_remove_dir(kobj);
@@ -157,12 +157,11 @@ static void unlink(struct kobject * kobj)
 }

 /**
- * kobject_shadow_add - add an object to the hierarchy.
+ * kobject_add - add an object to the hierarchy.
  * @kobj: object.
- * @shadow_parent: sysfs directory to add to.
  */

-int kobject_shadow_add(struct kobject *kobj, struct sysfs_dirent *shadow_parent)
+int kobject_add(struct kobject * kobj)
 {
  int error = 0;
  struct kobject * parent;
@@ -194,7 +193,7 @@ int kobject_shadow_add(struct kobject *kobj, struct sysfs_dirent
*shadow_parent)
   kobj->parent = parent;
  }

- error = create_dir(kobj, shadow_parent);
+ error = create_dir(kobj);
  if (error) {
```

```
   /* unlink does the kobject_put() for us */
   unlink(kobj);
@@ -216,16 +215,6 @@ int kobject_shadow_add(struct kobject *kobj, struct sysfs_dirent
*shadow_parent)
 }

 /**
- * kobject_add - add an object to the hierarchy.
- * @kobj: object.
- */
-int kobject_add(struct kobject * kobj)
-{
- return kobject_shadow_add(kobj, NULL);
-}
-
-
-/**
  * kobject_register - initialize and add an object.
  * @kobj: object in question.
  */
@@ -338,7 +327,7 @@ int kobject_rename(struct kobject * kobj, const char *new_name)
  /* Note : if we want to send the new name alone, not the full path,
   * we could probably use kobject_name(kobj); */

- error = sysfs_rename_dir(kobj, kobj->parent->sd, new_name);
+ error = sysfs_rename_dir(kobj, new_name);

  /* This function is mostly/only used for network interface.
   * Some hotplug package track interfaces by their name and
@@ -355,27 +344,6 @@ out:
 }

 /**
- * kobject_rename - change the name of an object
- * @kobj: object in question.
- * @new_parent: object's new parent
- * @new_name: object's new name
- */
-
-int kobject_shadow_rename(struct kobject *kobj,
-     struct sysfs_dirent *new_parent, const char *new_name)
-{
- int error = 0;
-
- kobj = kobject_get(kobj);
- if (!kobj)
-  return -EINVAL;
- error = sysfs_rename_dir(kobj, new_parent, new_name);
```

```
- kobject_put(kobj);
-
- return error;
-}
-
-/**
  * kobject_move - move object to another parent
  * @kobj: object in question.
  * @new_parent: object's new parent (can be NULL)
--
1.5.1.1.181.g2de0
```



---

## Subject: [PATCH 02/14] sysfs: In sysfs_lookup protect s_parent with sysfs_mutex
Posted by ebiederm on Tue, 31 Jul 2007 10:20:05 GMT
View Forum Message <> Reply to Message

It turns out we have been being silly and have failed to
protect ourselves against races between sysfs_lookup
and operations that modify the sysfs directory like sysfs_readdir
and create_dir.

So this patch modifies sysfs_lookup to grab sysfs_mutex before
walking the parent->s_children list.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/dir.c |   15 ++++++++------
 1 files changed, 9 insertions(+), 6 deletions(-)

```
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 40b9efe..b27a38c 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -756,11 +756,13 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry
*dentry,
     struct nameidata *nd)
 {
  struct sysfs_dirent * parent_sd = dentry->d_parent->d_fsdata;
+ struct dentry *result = NULL;
  struct sysfs_dirent * sd;
  struct bin_attribute *bin_attr;
  struct inode *inode;
```

```
  int found = 0;

+ 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)) {
@@ -771,14 +773,14 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry
*dentry,

  /* no such entry */
  if (!found)
-  return NULL;
+  goto out;

  /* attach dentry and inode */
  inode = sysfs_get_inode(sd);
- if (!inode)
-  return ERR_PTR(-ENOMEM);
-
- mutex_lock(&sysfs_mutex);
+ if (!inode) {
+  result = ERR_PTR(-ENOMEM);
+  goto out;
+ }

  if (inode->i_state & I_NEW) {
   /* initialize inode according to type */
@@ -808,9 +810,10 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry
*dentry,
  sysfs_instantiate(dentry, inode);
  sysfs_attach_dentry(sd, dentry);

+out:
  mutex_unlock(&sysfs_mutex);

- return NULL;
+ return result;
 }

 const struct inode_operations sysfs_dir_inode_operations = {
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 03/14] sysfs: Move all of inode initialization into sysfs_init_inode
Posted by ebiederm on Tue, 31 Jul 2007 10:22:56 GMT
View Forum Message <> Reply to Message

Besides being a good general cleanup the ultimate effect
of this patch is that sysfs_get_inode now gives you a
usable inode that doesn't need any further initialization
to be useful.

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 b27a38c..834ed74 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -741,24 +741,12 @@ 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 sysfs_dirent * parent_sd = dentry->d_parent->d_fsdata;
  struct dentry *result = NULL;
  struct sysfs_dirent * sd;
- struct bin_attribute *bin_attr;
  struct inode *inode;
  int found = 0;

@@ -782,31 +770,6 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry
*dentry,
   goto out;

```
  }

- 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 9671164..e832a5d 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 04/14] sysfs: Remove unnecessary variable found from sysfs_lookup

Posted by ebiederm on Tue, 31 Jul 2007 10:24:08 GMT

View Forum Message <> Reply to Message

The subject says it all.

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

```
---
 fs/sysfs/dir.c |    7 ++-----
 1 files changed, 2 insertions(+), 5 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 834ed74..8430633 100644
```

```
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -748,19 +748,16 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry
*dentry,
  struct dentry *result = NULL;
  struct sysfs_dirent * sd;
  struct inode *inode;
- int found = 0;

  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)) {
-   found = 1;
+      !strcmp(sd->s_name, dentry->d_name.name))
    break;
-  }
  }

  /* no such entry */
- if (!found)
+ if (!sd)
   goto out;

  /* attach dentry and inode */
--
1.5.1.1.181.g2de0
```



---

## Subject: [PATCH 05/14] sysfs: Remove sysfs_instantiate
Posted by ebiederm on Tue, 31 Jul 2007 10:25:26 GMT
View Forum Message <> Reply to Message

Now that sysfs_get_inode is dropping the inode lock
we no longer have a need for sysfs_instantiate.  So
kill it and just use d_instantiate instead.

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 8430633..6933f36 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -767,7 +767,7 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry *dentry,
   goto out;
  }

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

 out:
diff --git a/fs/sysfs/inode.c b/fs/sysfs/inode.c
index e832a5d..f55d9da 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 b55e510..3b4b989 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -71,7 +71,6 @@ extern void sysfs_remove_one(struct sysfs_addrm_cxt *acxt,
 extern int 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 06/14] sysfs: Rewrite sysfs_get_dentry
Posted by ebiederm on Tue, 31 Jul 2007 10:27:17 GMT
View Forum Message <> Reply to Message

Currently sysfs_get_dentry is very hairy and requires all kinds
of locking magic.  This patch rewrites sysfs_get_dentry to
not use the cached sd->s_dentry, and instead simply lookup
and create dcache entries.

This lays the foundation for removing s_dentry and the
current hairy sysfs_assoc_lock logic.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/dir.c |  117 ++++++++++++++++++++++++++++++++----------------------
 1 files changed, 49 insertions(+), 68 deletions(-)

```
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 6933f36..f22d60c 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -20,6 +20,8 @@ spinlock_t sysfs_assoc_lock = SPIN_LOCK_UNLOCKED;
 static spinlock_t sysfs_ino_lock = SPIN_LOCK_UNLOCKED;
 static DEFINE_IDA(sysfs_ino_ida);

+static void sysfs_attach_dentry(struct sysfs_dirent *sd, struct dentry *dentry);
+
 /**
  * sysfs_link_sibling - link sysfs_dirent into sibling list
  * @sd: sysfs_dirent of interest
@@ -66,10 +68,10 @@ 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.
+ * Get dentry for @sd.
+ *
+ * This function descends the sysfs dentry tree from the root
+ * populating it if necessary until it reaches the dentry for @sd.
  *
  * LOCKING:
  * Kernel thread context (may sleep)
@@ -77,85 +79,58 @@ void sysfs_unlink_sibling(struct sysfs_dirent *sd)
  * RETURNS:
  * Pointer to found dentry on success, ERR_PTR() value on error.
  */
-struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd)
+struct dentry *__sysfs_get_dentry(struct sysfs_dirent *sd, int create)
 {
  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);
+ struct qstr name;
+ struct inode *inode;
```

```
- spin_unlock(&dcache_lock);
- spin_unlock(&sysfs_assoc_lock);
+ parent_dentry = NULL;
+ dentry = dget(sysfs_sb->s_root);

- /* 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++)
+ do {
+  /* 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 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;
+  parent_dentry = dentry;
+  name.name = cur->s_name;
+  name.len = strlen(cur->s_name);
+  dentry = d_hash_and_lookup(parent_dentry, &name);
+  if (dentry)
+   continue;
+  if (!create)
+   goto out;
+  dentry = d_alloc(parent_dentry, &name);
+  if (!dentry) {
+   dentry = ERR_PTR(-ENOMEM);
```

```
+   goto out;
   }
-
-  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) {
+  inode = sysfs_get_inode(cur);
+  if (!inode) {
    dput(dentry);
-   sysfs_put(cur);
-   goto restart1;
+   dentry = ERR_PTR(-ENOMEM);
+   goto out;
   }
+  d_instantiate(dentry, inode);
+  sysfs_attach_dentry(cur, dentry);
+ } while (cur != sd);

-  spin_unlock(&sysfs_assoc_lock);
+out:
+ dput(parent_dentry);
+ return dentry;
+}

-  sysfs_put(cur);
- }
+struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd)
+{
+ struct dentry *dentry;

+ mutex_lock(&sysfs_mutex);
+ dentry = __sysfs_get_dentry(sd, 1);
  mutex_unlock(&sysfs_mutex);
  return dentry;
 }
@@ -750,6 +725,12 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry
*dentry,
  struct inode *inode;

  mutex_lock(&sysfs_mutex);
+
+ /* Guard against races with sysfs_get_dentry */
+ result = d_hash_and_lookup(dentry->d_parent, &dentry->d_name);
+ if (result)
```

```
+  goto out;
+
  for (sd = parent_sd->s_children; sd; sd = sd->s_sibling) {
   if (sysfs_type(sd) &&
       !strcmp(sd->s_name, dentry->d_name.name))
--
1.5.1.1.181.g2de0
```



---

Subject: [PATCH 07/14] vfs: Remove lookup_one_len_kern
Posted by ebiederm on Tue, 31 Jul 2007 10:28:15 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.

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 08/14] sysfs: Perform renames under sysfs_mutex
Posted by ebiederm on Tue, 31 Jul 2007 10:29:46 GMT
View Forum Message <> Reply to Message

This modifies sysfs_rename_dir to use sysfs_addrm_start/sysfs_addrm_finish
making it more like the rest of the sysfs directory manipulation functions.

This moves the entire rename operation inside of sysfs_mutex removing
the need to grab i_mutex, and incidentally this kills one s_dentry use.

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

```
 fs/sysfs/dir.c |   46 ++++++++++++++++++++++++++++------------------
 1 files changed, 17 insertions(+), 29 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index f22d60c..8e614d3 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -827,7 +827,7 @@ void sysfs_remove_dir(struct kobject * kobj)
 int sysfs_rename_dir(struct kobject *kobj, const char *new_name)
 {
  struct sysfs_dirent *sd = kobj->sd;
- struct dentry *new_parent = NULL;
+ struct sysfs_addrm_cxt acxt;
  struct dentry *old_dentry = NULL, *new_dentry = NULL;
  const char *dup_name = NULL;
  int error;
@@ -836,59 +836,47 @@ int sysfs_rename_dir(struct kobject *kobj, const char *new_name)
  old_dentry = sysfs_get_dentry(sd);
  if (IS_ERR(old_dentry)) {
   error = PTR_ERR(old_dentry);
-  goto out_dput;
+  goto out;
  }

- new_parent = dget(old_dentry->d_parent);
-
- /* lock new_parent and get dentry for new name */
- mutex_lock(&new_parent->d_inode->i_mutex);
-
- new_dentry = lookup_one_len(new_name, new_parent, strlen(new_name));
- if (IS_ERR(new_dentry)) {
-  error = PTR_ERR(new_dentry);
-  goto out_unlock;
- }
+ sysfs_addrm_start(&acxt, sd->s_parent);

  error = -EINVAL;
- if (old_dentry == new_dentry)
-  goto out_unlock;
+ if (strcmp(new_name, sd->s_name) == 0)
+  goto addrm_finish;

  error = -EEXIST;
- if (new_dentry->d_inode)
-  goto out_unlock;
+ if (sysfs_find_dirent(acxt.parent_sd, new_name))
+  goto addrm_finish;
+
```

```
+ new_dentry = d_alloc_name(old_dentry->d_parent, new_name);
+ if (!new_dentry)
+  goto addrm_finish;

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

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

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

  /* move under the new parent */
  d_add(new_dentry, NULL);
- d_move(sd->s_dentry, new_dentry);
+ d_move(old_dentry, new_dentry);

  error = 0;
- goto out_unlock;
-
- out_drop:
- d_drop(new_dentry);
- out_unlock:
- mutex_unlock(&new_parent->d_inode->i_mutex);
- out_dput:
+addrm_finish:
+ sysfs_addrm_finish(&acxt);
  kfree(dup_name);
- dput(new_parent);
  dput(old_dentry);
  dput(new_dentry);
+out:
  return error;
 }

--
1.5.1.1.181.g2de0
```

---

Subject: [PATCH 09/14] sysfs: Move all of sysfs_move_dir under sysfs_mutex
Posted by ebiederm on Tue, 31 Jul 2007 10:33:36 GMT
View Forum Message <> Reply to Message

This patch modifies sysfs_move_dir to perform all of it's
operations under the sysfs_mutex. By looking for conflicts
using sysfs_find_dirent we accidentally moving something
onto a name that already exists but just not in the dcache
right now.

Two s_dentry usages are killed.

And it has probably become unnecessary to grab i_mutex at
all but I'm not brave enough to do that just yet.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
fs/sysfs/dir.c | 24 ++++++++++++++----------
1 files changed, 13 insertions(+), 11 deletions(-)

```
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 8e614d3..ed2e6f3 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -897,7 +897,7 @@ 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)) {
@@ -915,29 +915,31 @@ again:
   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, kobj->name))
   goto out_unlock;
- } else
-  error = 0;
+
+ error = -ENOMEM;
+ new_dentry = d_alloc_name(new_parent, kobj->name);
+ if (!new_dentry)
+  goto out_unlock;
+
+ 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. */
- 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:
--
1.5.1.1.181.g2de0
```



---

## Subject: Re: [PATCH 02/14] sysfs: In sysfs_lookup protect s_parent with sysfs_mutex

Posted by Tejun Heo on Tue, 31 Jul 2007 10:36:39 GMT

View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> It turns out we have been being silly and have failed to

> protect ourselves against races between sysfs_lookup
> and operations that modify the sysfs directory like sysfs_readdir
> and create_dir.
>
> So this patch modifies sysfs_lookup to grab sysfs_mutex before
> walking the parent->s_children list.

Updated portion of #01 and #02 are identical to the cleanup patches I
just posted.  :-)

--
tejun

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

---

Subject: [PATCH 10/14] sysfs: Rework sysfs_drop_dentry
Posted by ebiederm on Tue, 31 Jul 2007 10:37:34 GMT
View Forum Message <> Reply to Message

This modifies sysfs_drop_dentry so it no longers needs the
sysfs_assoc_lock and it's assorted challenges.

The principal idea is that __sysfs_get_dentry when not creating dentry
will return a dentry for the sysfs_dirent if there is one in the
dcache.  If we do get a dentry we can drop it and force it out
of the cache.

For good measure I am also calling shrink_dcache_parent which will
force all of the child dentries out of the dcache as well if it
is a directory.

This is essentially and adaption of proc_flush_task from procfs
to handle the similar problems of sysfs.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/dir.c |   34 ++++++++++++----------------------
 1 files changed, 12 insertions(+), 22 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index ed2e6f3..f8de6fb 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -516,35 +516,25 @@ void sysfs_remove_one(struct sysfs_addrm_cxt *acxt, struct
sysfs_dirent *sd)

```
  * 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.
+ * We find the dentry for @sd by probing the dcache while holding
+ * sysfs_mutex to keep it from changing until we are done.
  */
 static void sysfs_drop_dentry(struct sysfs_dirent *sd)
 {
  struct dentry *dentry = NULL;
  struct inode *inode;

- /* We're not holding a reference to ->s_dentry dentry but the
-  * field will stay valid as long as sysfs_assoc_lock is held.
-  */
- spin_lock(&sysfs_assoc_lock);
- 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);
-  spin_lock(&dentry->d_lock);
-  __d_drop(dentry);
-  spin_unlock(&dentry->d_lock);
+ /* Remove a dentry for a sd from the dcache if present */
+ mutex_lock(&sysfs_mutex);
+ dentry = __sysfs_get_dentry(sd, 0);
+ if (IS_ERR(dentry))
+  dentry = NULL;
+ if (dentry) {
+  shrink_dcache_parent(dentry);
+  d_drop(dentry);
+  dput(dentry);
  }
-
- spin_unlock(&dcache_lock);
- spin_unlock(&sysfs_assoc_lock);
-
- dput(dentry);
+ mutex_unlock(&sysfs_mutex);

  /* adjust nlink and update timestamp */
  inode = ilookup(sysfs_sb, sd->s_ino);
--
```

1.5.1.1.181.g2de0



Subject: [PATCH 11/14] sysfs: Remove s_dentry form sysfs_dirent.
Posted by ebiederm on Tue, 31 Jul 2007 10:39:05 GMT
View Forum Message <> Reply to Message

After the last round of refactoring nothing is really using
using s_dirent so remove it. Which makes the code easier to work
with and follow. And with a little luck makes it easier to
support multiple dentry trees for sysfs.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
fs/sysfs/dir.c | 26 +------------------------
fs/sysfs/mount.c | 1 -
fs/sysfs/sysfs.h | 1 -
3 files changed, 1 insertions(+), 27 deletions(-)

```
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index f8de6fb..c17e601 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -310,22 +310,7 @@ 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);
 }

@@ -373,9 +358,6 @@ 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)
  */
@@ -383,12 +365,6 @@ 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);
 }

diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index 919eaaf..3f652be 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 3b4b989..60af864 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 12/14] sysfs: Make sysfs_mount static
Posted by ebiederm on Tue, 31 Jul 2007 10:40:25 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 clearer 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 c17e601..1611cbd 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -671,10 +671,8 @@ 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 3f652be..0ad731b 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 4ce687f..2b542dc 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 60af864..265a16a 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -50,7 +50,6 @@ 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 13/14] sysfs: Simplify readdir.
Posted by ebiederm on Tue, 31 Jul 2007 10:42:02 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 achived 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 removal 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 | 178 ++++++++++++++----------------------------
fs/sysfs/inode.c | 2 -
2 files changed, 45 insertions(+), 135 deletions(-)

```
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 1611cbd..f9be856 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -35,10 +35,20 @@ static void sysfs_attach_dentry(struct sysfs_dirent *sd, struct dentry
*dentry);
 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;
 }

 /**
@@ -590,7 +600,7 @@ 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;
 }
@@ -696,8 +706,7 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry *dentry,
   goto out;

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

@@ -756,7 +765,7 @@ 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) {
    *pos = sd->s_sibling;
    sd->s_sibling = NULL;
    sysfs_remove_one(&acxt, sd);
@@ -913,37 +922,6 @@ 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)
 {
@@ -954,117 +932,51 @@ 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,
 };
diff --git a/fs/sysfs/inode.c b/fs/sysfs/inode.c
index f55d9da..e212c69 100644
--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -225,8 +225,6 @@ int sysfs_hash_and_remove(struct sysfs_dirent *dir_sd, const char
*name)
  for (pos = &dir_sd->s_children; *pos; pos = &(*pos)->s_sibling) {
   sd = *pos;

-  if (!sysfs_type(sd))
-   continue;
   if (!strcmp(sd->s_name, name)) {
    *pos = sd->s_sibling;
    sd->s_sibling = NULL;
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 14/14] sysfs: In sysfs_lookup don't open code sysfs_find_dirent
Posted by ebiederm on Tue, 31 Jul 2007 10:42:47 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 |    5 +----
 1 files changed, 1 insertions(+), 4 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index f9be856..66d418a 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -705,10 +705,7 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry
*dentry,
  if (result)
   goto out;

- for (sd = parent_sd->s_children; sd; sd = sd->s_sibling) {
-  if (!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



Subject: Re: [PATCH 03/14] sysfs: Move all of inode initialization into sysfs_init_inode

Posted by Tejun Heo on Tue, 31 Jul 2007 10:43:40 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Besides being a good general cleanup the ultimate effect
> of this patch is that sysfs_get_inode now gives you a
> usable inode that doesn't need any further initialization
> to be useful.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

I generally like it. The only nits are that I think we're better off keeping the function in fs/sysfs/dir.c as all the interesting parts are there anyway and sysfs_instantiate() can just die. The function is just a remnant of days where inode initialization was scattered all over the place.

Thanks.

--
tejun

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

Subject: Re: [PATCH 05/14] sysfs: Remove sysfs_instantiate
Posted by Tejun Heo on Tue, 31 Jul 2007 10:44:42 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Now that sysfs_get_inode is dropping the inode lock
> we no longer have a need for sysfs_instantiate. So
> kill it and just use d_instantiate instead.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Oh, here it is. Forget the second nit in my previous comment. Yeap, this looks good.

--
tejun

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

Subject: Re: [PATCH 04/14] sysfs: Remove unnecessary variable found from sysfs_lookup
Posted by Tejun Heo on Tue, 31 Jul 2007 10:45:20 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> The subject says it all.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

I happen to have just posted the same patch. :-)

--
tejun

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

Subject: Re: [PATCH 02/14] sysfs: In sysfs_lookup protect s_parent with sysfs_mutex
Posted by ebiederm on Tue, 31 Jul 2007 10:45:21 GMT
View Forum Message <> Reply to Message

Tejun Heo <teheo@suse.de> writes:

> Eric W. Biederman wrote:
>> It turns out we have been being silly and have failed to
>> protect ourselves against races between sysfs_lookup
>> and operations that modify the sysfs directory like sysfs_readdir
>> and create_dir.
>>
>> So this patch modifies sysfs_lookup to grab sysfs_mutex before
>> walking the parent->s_children list.
>
> Updated portion of #01 and #02 are identical to the cleanup patches I
> just posted. :-)

I'm not surprised. I will have to see what you have done after I have
slept.

Eric

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

Subject: Re: [PATCH 02/14] sysfs: In sysfs_lookup protect s_parent with sysfs_mutex
Posted by Tejun Heo on Tue, 31 Jul 2007 10:47:56 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Tejun Heo <teheo@suse.de> writes:
>
>> Eric W. Biederman wrote:
>>> It turns out we have been being silly and have failed to
>>> protect ourselves against races between sysfs_lookup
>>> and operations that modify the sysfs directory like sysfs_readdir
>>> and create_dir.
>>>
>>> So this patch modifies sysfs_lookup to grab sysfs_mutex before
>>> walking the parent->s_children list.
>> Updated portion of #01 and #02 are identical to the cleanup patches I
>> just posted. :-)
>
> I'm not surprised. I will have to see what you have done after I have
> slept.

Good night.

--
tejun

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

Subject: Re: [PATCH 06/14] sysfs: Rewrite sysfs_get_dentry
Posted by Tejun Heo on Tue, 31 Jul 2007 10:59:02 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Currently sysfs_get_dentry is very hairy and requires all kinds
> of locking magic. This patch rewrites sysfs_get_dentry to
> not use the cached sd->s_dentry, and instead simply lookup
> and create dcache entries.

I wanted to kill sd->s_dentry from the beginning. The obstacle was
actually the shadow directory support, ironic.

> + struct qstr name;
> + struct inode *inode;
>

> + parent_dentry = NULL;
> + dentry = dget(sysfs_sb->s_root);
>
> + do {
> +  /* 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 */
>    dput(parent_dentry);

Shouldn't this be done after looking up the child?

> +  parent_dentry = dentry;
> +  name.name = cur->s_name;
> +  name.len = strlen(cur->s_name);
> +  dentry = d_hash_and_lookup(parent_dentry, &name);
> +  if (dentry)
> +   continue;
> +  if (!create)
> +   goto out;

Probably missing dentry = NULL?

> +  dentry = d_alloc(parent_dentry, &name);
> +  if (!dentry) {
> +   dentry = ERR_PTR(-ENOMEM);
> +   goto out;
>    }
> +  inode = sysfs_get_inode(cur);
> +  if (!inode) {
>     dput(dentry);
> +   dentry = ERR_PTR(-ENOMEM);
> +   goto out;
>    }
> +  d_instantiate(dentry, inode);
> +  sysfs_attach_dentry(cur, dentry);
> + } while (cur != sd);
>
> +out:
> + dput(parent_dentry);
> + return dentry;
> +}
>
> @@ -750,6 +725,12 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry
*dentry,
>   struct inode *inode;

>
>  mutex_lock(&sysfs_mutex);
> +
> + /* Guard against races with sysfs_get_dentry */
> + result = d_hash_and_lookup(dentry->d_parent, &dentry->d_name);
> + if (result)
> +  goto out;
> +

Hmmm... This is tricky but probably better than the previous hairy
sysfs_get_dentry().  I think it would be worthwhile to comment about
creating dentry/inode behind vfs's back in __sysfs_get_dentry().

One thing I'm worried about is that dentry can now be looked up without
holding i_mutex.  In sysfs proper, there's no synchronization problem
but I'm not sure whether we're willing to cross that line set by vfs.
It might come back and bite our asses hard later.

Thanks.

--
tejun

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

---

Subject: Re: [PATCH 08/14] sysfs: Perform renames under sysfs_mutex
Posted by Tejun Heo on Tue, 31 Jul 2007 11:06:09 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> This modifies sysfs_rename_dir to use sysfs_addrm_start/sysfs_addrm_finish
> making it more like the rest of the sysfs directory manipulation functions.
>
> This moves the entire rename operation inside of sysfs_mutex removing
> the need to grab i_mutex, and incidentally this kills one s_dentry use.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Looks good.

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

--
tejun

_____

Subject: Re: [PATCH 09/14] sysfs: Move all of sysfs_move_dir under sysfs_mutex
Posted by Tejun Heo on Tue, 31 Jul 2007 11:09:12 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> This patch modifies sysfs_move_dir to perform all of it's
> operations under the sysfs_mutex.  By looking for conflicts
> using sysfs_find_dirent we accidentally moving something
> onto a name that already exists but just not in the dcache
> right now.
>
> Two s_dentry usages are killed.
>
> And it has probably become unnecessary to grab i_mutex at
> all but I'm not brave enough to do that just yet.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

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

--
tejun



Subject: Re: [PATCH 08/14] sysfs: Perform renames under sysfs_mutex
Posted by Tejun Heo on Tue, 31 Jul 2007 11:10:59 GMT
View Forum Message <> Reply to Message

Tejun Heo wrote:
> Eric W. Biederman wrote:
>> This modifies sysfs_rename_dir to use sysfs_addrm_start/sysfs_addrm_finish
>> making it more like the rest of the sysfs directory manipulation functions.
>>
>> This moves the entire rename operation inside of sysfs_mutex removing
>> the need to grab i_mutex, and incidentally this kills one s_dentry use.
>>
>> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
>

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

Just one afterthought.  addrm_start/finish are basically used as wrapper
to lock both sysfs_mutex and i_mutex.  It might be better to just open
code them.  sysfs_move_dir() has to open code locking anyway and IMHO
it's better to keep rename and move look similar.

Thanks.

--
tejun



---

Subject: Re: [PATCH 10/14] sysfs: Rework sysfs_drop_dentry
Posted by Tejun Heo on Tue, 31 Jul 2007 11:17:57 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> This modifies sysfs_drop_dentry so it no longers needs the
> sysfs_assoc_lock and it's assorted challenges.
>
> The principal idea is that __sysfs_get_dentry when not creating dentry
> will return a dentry for the sysfs_dirent if there is one in the
> dcache.  If we do get a dentry we can drop it and force it out
> of the cache.
>
> For good measure I am also calling shrink_dcache_parent which will
> force all of the child dentries out of the dcache as well if it
> is a directory.

I don't think we need this as we drop all sd's recursively anyway.  Can
we drop it?

Other than that, looks good.

--
tejun

Subject: Re: [PATCH 11/14] sysfs: Remove s_dentry form sysfs_dirent.
Posted by Tejun Heo on Tue, 31 Jul 2007 11:18:22 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> After the last round of refactoring nothing is really using
> using s_dirent so remove it.  Which makes the code easier to work
> with and follow.  And with a little luck makes it easier to
> support multiple dentry trees for sysfs.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

Hooray, 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 12/14] sysfs: Make sysfs_mount static
Posted by Tejun Heo on Tue, 31 Jul 2007 11:19:51 GMT
View Forum Message <> Reply to Message

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 clearer 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 06/14] sysfs: Rewrite sysfs_get_dentry
Posted by ebiederm on Tue, 31 Jul 2007 11:23:58 GMT

Tejun Heo <teheo@suse.de> writes:

> Eric W. Biederman wrote:
>> Currently sysfs_get_dentry is very hairy and requires all kinds
>> of locking magic.  This patch rewrites sysfs_get_dentry to
>> not use the cached sd->s_dentry, and instead simply lookup
>> and create dcache entries.
>
> I wanted to kill sd->s_dentry from the beginning.  The obstacle was
> actually the shadow directory support, ironic.
>
>> + struct qstr name;
>> + struct inode *inode;
>>
>> + parent_dentry = NULL;
>> + dentry = dget(sysfs_sb->s_root);
>>
>> + do {
>> +  /* 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 */
>>    dput(parent_dentry);
>
> Shouldn't this be done after looking up the child?
Yes and that is what this is.  Delaying it a little longer
made the conditionals easier to write and verify for correctness.

>> +  parent_dentry = dentry;
>> +  name.name = cur->s_name;
>> +  name.len = strlen(cur->s_name);
>> +  dentry = d_hash_and_lookup(parent_dentry, &name);
>> +  if (dentry)
>> +   continue;
>> +  if (!create)
>> +   goto out;
>
> Probably missing dentry = NULL?
d_hash_and_lookup sets dentry, and we can't get here if (dentry != NULL)

>> +  dentry = d_alloc(parent_dentry, &name);
>> +  if (!dentry) {
>> +   dentry = ERR_PTR(-ENOMEM);
>> +   goto out;
>>    }

>> +  inode = sysfs_get_inode(cur);
>> +  if (!inode) {
>>     dput(dentry);
>> +   dentry = ERR_PTR(-ENOMEM);
>> +   goto out;
>>    }
>> +  d_instantiate(dentry, inode);
>> +  sysfs_attach_dentry(cur, dentry);
>> + } while (cur != sd);
>>
>> +out:
>> + dput(parent_dentry);
>> + return dentry;
>> +}
>>
>> @@ -750,6 +725,12 @@ static struct dentry * sysfs_lookup(struct inode *dir,
> struct dentry *dentry,
>>   struct inode *inode;
>>
>>   mutex_lock(&sysfs_mutex);
>> +
>> + /* Guard against races with sysfs_get_dentry */
>> + result = d_hash_and_lookup(dentry->d_parent, &dentry->d_name);
>> + if (result)
>> +  goto out;
>> +
>
> Hmmm... This is tricky but probably better than the previous hairy
> sysfs_get_dentry().  I think it would be worthwhile to comment about
> creating dentry/inode behind vfs's back in __sysfs_get_dentry().

Yes. Good point.  That is sufficiently non-intuitive and non-obvious
to deserve a comment.

> One thing I'm worried about is that dentry can now be looked up without
> holding i_mutex.  In sysfs proper, there's no synchronization problem
> but I'm not sure whether we're willing to cross that line set by vfs.
> It might come back and bite our asses hard later.

It's a reasonable concern.  I'm wondering if there are any precedents
set by distributed filesystems.  Because in a sense that is what
we are.

As for crossing that line I don't know what to say it makes the
code a lot cleaner, and if we are merged into the kernel at
least it will be visible if someone rewrites the vfs.

If sysfs_mutex nested the other way things would be easier,

and we could grab all of the i_mutexes we wanted.  I wonder if we can
be annoying in sysfs_lookup and treat that as the lock inversion
case using mutex_trylock etc.  And have sysfs_mutex be on the
outside for the rest of the cases?

Anyway back to bed with me.  I've been dreaming up to many silly
ways to abuse the dcache...

Eric

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

---

Subject: Re: [PATCH 06/14] sysfs: Rewrite sysfs_get_dentry
Posted by Tejun Heo on Tue, 31 Jul 2007 11:34:47 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
>>> + do {
>>> +  /* 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 */
>>>    dput(parent_dentry);
>> Shouldn't this be done after looking up the child?
> Yes and that is what this is.  Delaying it a little longer
> made the conditionals easier to write and verify for correctness.

Right, missed the next line.

>>> +  parent_dentry = dentry;
>>> +  name.name = cur->s_name;
>>> +  name.len = strlen(cur->s_name);
>>> +  dentry = d_hash_and_lookup(parent_dentry, &name);
>>> +  if (dentry)
>>> +   continue;
>>> +  if (!create)
>>> +   goto out;
>> Probably missing dentry = NULL?
> d_hash_and_lookup sets dentry, and we can't get here if (dentry != NULL)

Yes.

>> One thing I'm worried about is that dentry can now be looked up without

>> holding i_mutex.  In sysfs proper, there's no synchronization problem
>> but I'm not sure whether we're willing to cross that line set by vfs.
>> It might come back and bite our asses hard later.
>
> It's a reasonable concern.  I'm wondering if there are any precedents
> set by distributed filesystems.  Because in a sense that is what
> we are.

Yeah, that's the weird thing about sysfs.  sysfs interface acts as a
different access point to the filesystem making it virtually distributed.

> As for crossing that line I don't know what to say it makes the
> code a lot cleaner, and if we are merged into the kernel at
> least it will be visible if someone rewrites the vfs.
>
> If sysfs_mutex nested the other way things would be easier,
> and we could grab all of the i_mutexes we wanted.  I wonder if we can
> be annoying in sysfs_lookup and treat that as the lock inversion
> case using mutex_trylock etc.  And have sysfs_mutex be on the
> outside for the rest of the cases?

The problem with treating sysfs_lookup as inversion case is that vfs
layer grabs i_mutex outside of sysfs_lookup.  Releasing i_mutex from
inside sysfs_lookup would be a hacky layering violation.

Then again, the clean up which can come from the new sysfs_looukp_dentry
is very significant.  I'll think about it a bit more.

Thanks.

--
tejun

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

---

Subject: Re: [PATCH 13/14] sysfs: Simplify readdir.
Posted by Tejun Heo on Tue, 31 Jul 2007 11:36:08 GMT
View Forum Message <> Reply to Message

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 achived 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 removal 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>

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

--
tejun



---

Subject: Re: [PATCH 14/14] sysfs: In sysfs_lookup don't open code sysfs_find_dirent
Posted by Tejun Heo on Tue, 31 Jul 2007 11:38:16 GMT
View Forum Message <> Reply to Message

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>

I think this series of patches need wider review than Linux Containers
mailing list.  Please cc LKML, cornelia.huck@de.ibm.com and
stern@rowland.harvard.edu when posting the next round.

Thanks.

--
tejun

Subject: Re: [PATCH 06/14] sysfs: Rewrite sysfs_get_dentry
Posted by Tejun Heo on Tue, 31 Jul 2007 14:16:13 GMT
View Forum Message <> Reply to Message

On Tue, Jul 31, 2007 at 08:34:47PM +0900, Tejun Heo wrote:
> > If sysfs_mutex nested the other way things would be easier,
> > and we could grab all of the i_mutexes we wanted. I wonder if we can
> > be annoying in sysfs_lookup and treat that as the lock inversion
> > case using mutex_trylock etc. And have sysfs_mutex be on the
> > outside for the rest of the cases?
>
> The problem with treating sysfs_lookup as inversion case is that vfs
> layer grabs i_mutex outside of sysfs_lookup. Releasing i_mutex from
> inside sysfs_lookup would be a hacky layering violation.
>
> Then again, the clean up which can come from the new sysfs_looukp_dentry
> is very significant. I'll think about it a bit more.

How about something like this? __sysfs_get_dentry() never creates any dentry, it just looks up existing ones. sysfs_get_dentry() calls __sysfs_get_dentry() and if it fails, it builds a path string and look up using regular vfs_path_lookup(). Once in the creation path, sysfs_get_dentry() is allowed to fail, so allocating path buf is fine.

It still needs to retry when vfs_path_lookup() returns -ENOENT or the wrong dentry but things are much simpler now. It doesn't violate any VFS locking rule while maintaining all the benefits of sysfs_get_dentry() cleanup.

Something like LOOKUP_KERNEL is needed to ignore security checks; otherwise, we'll need to resurrect lookup_one_len_kern() and open code look up.

The patch is on top of all your patches and is in barely working form.

```
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 66d418a..0a6e036 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -81,20 +81,19 @@ void sysfs_unlink_sibling(struct sysfs_dirent *sd)
  * Get dentry for @sd.
  *
  * This function descends the sysfs dentry tree from the root
- * populating it if necessary until it reaches the dentry for @sd.
+ * until it reaches the dentry for @sd.
  *
  * LOCKING:
- * Kernel thread context (may sleep)
+ * mutex_lock(sysfs_mutex)
```

```
  *
  * RETURNS:
- * Pointer to found dentry on success, ERR_PTR() value on error.
+ * Pointer to found dentry on success, NULL if doesn't exist.
  */
-struct dentry *__sysfs_get_dentry(struct sysfs_dirent *sd, int create)
+struct dentry *__sysfs_get_dentry(struct sysfs_dirent *sd)
 {
  struct sysfs_dirent *cur;
  struct dentry *parent_dentry, *dentry;
  struct qstr name;
- struct inode *inode;

  parent_dentry = NULL;
  dentry = dget(sysfs_sb->s_root);
@@ -111,26 +110,8 @@ struct dentry *__sysfs_get_dentry(struct sysfs_dirent *sd, int create)
   name.name = cur->s_name;
   name.len = strlen(cur->s_name);
   dentry = d_hash_and_lookup(parent_dentry, &name);
-  if (dentry)
-   continue;
-  if (!create)
-   goto out;
-  dentry = d_alloc(parent_dentry, &name);
-  if (!dentry) {
-   dentry = ERR_PTR(-ENOMEM);
-   goto out;
-  }
-  inode = sysfs_get_inode(cur);
-  if (!inode) {
-   dput(dentry);
-   dentry = ERR_PTR(-ENOMEM);
-   goto out;
-  }
-  d_instantiate(dentry, inode);
-  sysfs_attach_dentry(cur, dentry);
- } while (cur != sd);
+ } while (dentry && cur != sd);

-out:
  dput(parent_dentry);
  return dentry;
 }
@@ -138,10 +119,52 @@ out:
 struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd)
 {
  struct dentry *dentry;
+ struct nameidata nd;
```

```
+ char *path_buf;
+ int len, rc;
+
+ mutex_lock(&sysfs_mutex);
+ dentry = __sysfs_get_dentry(sd);
+ mutex_unlock(&sysfs_mutex);
+
+ if (dentry)
+  return dentry;
+
+ /* Look up failed.  This means that the dentry associated with
+  * @sd currently doesn't exist and we're allowed to fail.
+  * Let's allocate path buffer and look up like a sane person.
+  */
+ path_buf = kmalloc(PATH_MAX, GFP_KERNEL);
+ if (!path_buf)
+  return ERR_PTR(-ENOMEM);

+ retry:
  mutex_lock(&sysfs_mutex);
- dentry = __sysfs_get_dentry(sd, 1);
+ /* XXX - clean up */
+ len = object_path_length(sd);
+ BUG_ON(len > PATH_MAX);
+ path_buf[len - 1] = '\0';
+ fill_object_path(sd, path_buf, len);
  mutex_unlock(&sysfs_mutex);
+
+ /* XXX - we need a flag to override security check or need to
+  * open code lookup.  The former is far better, IMHO.
+  */
+ rc = vfs_path_lookup(sysfs_mount->mnt_root, sysfs_mount,
+        path_buf + 1, 0, &nd);
+ dentry = nd.dentry;
+
+ /* renamed/moved beneath us? */
+ if (rc == -ENOENT)
+  goto retry;
+ if (rc == 0 && dentry->d_fsdata != sd) {
+  dput(dentry);
+  goto retry;
+ }
+ if (rc && rc != -ENOENT)
+  dentry = ERR_PTR(rc);
+
+ kfree(path_buf);
  return dentry;
 }
```

```
@@ -512,7 +535,7 @@ static void sysfs_drop_dentry(struct sysfs_dirent *sd)

  /* Remove a dentry for a sd from the dcache if present */
  mutex_lock(&sysfs_mutex);
- dentry = __sysfs_get_dentry(sd, 0);
+ dentry = __sysfs_get_dentry(sd);
  if (IS_ERR(dentry))
   dentry = NULL;
  if (dentry) {
@@ -700,11 +723,6 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry
*dentry,

  mutex_lock(&sysfs_mutex);

- /* Guard against races with sysfs_get_dentry */
- result = d_hash_and_lookup(dentry->d_parent, &dentry->d_name);
- if (result)
-  goto out;
-
  sd = sysfs_find_dirent(parent_sd, dentry->d_name.name);

  /* no such entry */
diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index 0ad731b..3f652be 100644
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -15,7 +15,7 @@
 /* Random magic number */
 #define SYSFS_MAGIC 0x62656572

-static struct vfsmount *sysfs_mount;
+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 2b542dc..336823c 100644
--- a/fs/sysfs/symlink.c
+++ b/fs/sysfs/symlink.c
@@ -21,7 +21,7 @@ static int object_depth(struct sysfs_dirent *sd)
  return depth;
 }

-static int object_path_length(struct sysfs_dirent * sd)
+int object_path_length(struct sysfs_dirent * sd)
 {
  int length = 1;
```

```
@@ -31,7 +31,7 @@ static int object_path_length(struct sysfs_dirent * sd)
  return length;
 }

-static void fill_object_path(struct sysfs_dirent *sd, char *buffer, int length)
+void fill_object_path(struct sysfs_dirent *sd, char *buffer, int length)
 {
  --length;
  for (; sd->s_parent; sd = sd->s_parent) {
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index 265a16a..86704ef 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -51,6 +51,7 @@ struct sysfs_addrm_cxt {
 };

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

 extern struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd);
@@ -89,6 +90,9 @@ extern void sysfs_remove_subdir(struct sysfs_dirent *sd);

 extern int sysfs_setattr(struct dentry *dentry, struct iattr *iattr);

+extern int object_path_length(struct sysfs_dirent * sd);
+extern void fill_object_path(struct sysfs_dirent *sd, char *buffer, int length);
+
 extern spinlock_t sysfs_assoc_lock;
 extern struct mutex sysfs_mutex;
 extern struct super_block * sysfs_sb;
```



---

## Subject: Re: [PATCH 14/14] sysfs: In sysfs_lookup don't open code sysfs_find_dirent

Posted by ebiederm on Tue, 31 Jul 2007 16:01:12 GMT

View Forum Message <> Reply to Message

Tejun Heo <teheo@suse.de> writes:

> 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>
>
> I think this series of patches need wider review than Linux Containers
> mailing list.  Please cc LKML, cornelia.huck@de.ibm.com and
> stern@rowland.harvard.edu when posting the next round.

Sure.  I was just Cc'ing those I could think of and I was tired.

Eric



---

Subject: Re: [PATCH 06/14] sysfs: Rewrite sysfs_get_dentry
Posted by ebiederm on Tue, 31 Jul 2007 19:24:37 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> On Tue, Jul 31, 2007 at 08:34:47PM +0900, Tejun Heo wrote:
>> > If sysfs_mutex nested the other way things would be easier,
>> > and we could grab all of the i_mutexes we wanted.  I wonder if we can
>> > be annoying in sysfs_lookup and treat that as the lock inversion
>> > case using mutex_trylock etc.  And have sysfs_mutex be on the
>> > outside for the rest of the cases?
>>
>> The problem with treating sysfs_lookup as inversion case is that vfs
>> layer grabs i_mutex outside of sysfs_lookup.  Releasing i_mutex from
>> inside sysfs_lookup would be a hacky layering violation.
>>
>> Then again, the clean up which can come from the new sysfs_loukp_dentry
>> is very significant.  I'll think about it a bit more.
>
> How about something like this?  __sysfs_get_dentry() never creates any
> dentry, it just looks up existing ones.  sysfs_get_dentry() calls
> __sysfs_get_dentry() and if it fails, it builds a path string and look
> up using regular vfs_path_lookup().  Once in the creation path,
> sysfs_get_dentry() is allowed to fail, so allocating path buf is fine.
>
> It still needs to retry when vfs_path_lookup() returns -ENOENT or the
> wrong dentry but things are much simpler now.  It doesn't violate any

> VFS locking rule while maintaining all the benefits of
> sysfs_get_dentry() cleanup.
>
> Something like LOOKUP_KERNEL is needed to ignore security checks;
> otherwise, we'll need to resurrect lookup_one_len_kern() and open code
> look up.
>
> The patch is on top of all your patches and is in barely working form.

Thanks. I need to look some more. I've got a case where __sysfs_get_dentry
called from sysfs_drop_dentry is not successfully walking of the sysfs_dirent
tree to the initial root directory. (This is with deleted sysfs_dirents but
that expected from the context).

I haven't yet had a chance to dig in and see what is going on yet.

Your patch doesn't touch that part of my logic so I don't know yet what
is going on.

Eric



---

Subject: Re: [PATCH 06/14] sysfs: Rewrite sysfs_get_dentry
Posted by ebiederm on Wed, 01 Aug 2007 09:22:40 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> On Tue, Jul 31, 2007 at 08:34:47PM +0900, Tejun Heo wrote:
>> > If sysfs_mutex nested the other way things would be easier,
>> > and we could grab all of the i_mutexes we wanted. I wonder if we can
>> > be annoying in sysfs_lookup and treat that as the lock inversion
>> > case using mutex_trylock etc. And have sysfs_mutex be on the
>> > outside for the rest of the cases?
>>
>> The problem with treating sysfs_lookup as inversion case is that vfs
>> layer grabs i_mutex outside of sysfs_lookup. Releasing i_mutex from
>> inside sysfs_lookup would be a hacky layering violation.
>>
>> Then again, the clean up which can come from the new sysfs_loookp_dentry
>> is very significant. I'll think about it a bit more.
>
> How about something like this? __sysfs_get_dentry() never creates any
> dentry, it just looks up existing ones. sysfs_get_dentry() calls

> __sysfs_get_dentry() and if it fails, it builds a path string and look
> up using regular vfs_path_lookup(). Once in the creation path,
> sysfs_get_dentry() is allowed to fail, so allocating path buf is fine.
>
> It still needs to retry when vfs_path_lookup() returns -ENOENT or the
> wrong dentry but things are much simpler now. It doesn't violate any
> VFS locking rule while maintaining all the benefits of
> sysfs_get_dentry() cleanup.
>
> Something like LOOKUP_KERNEL is needed to ignore security checks;
> otherwise, we'll need to resurrect lookup_one_len_kern() and open code
> look up.
>
> The patch is on top of all your patches and is in barely working form.

I will look a little more and see. But right now it looks like the
real problem with locking is that we use sysfs_mutex to lock the
sysfs_dirent s_children list.

Instead it really looks like we should use i_mutex from the appropriate
inode. Or is there a real performance problem with forcing the directory
inodes in core when we modify the directories?

Using i_mutex to lock the s_children list. Allows us to make sysfs_mutex
come before i_mutex, and it removes the need for an additional lock in
sysfs_lookup. So generally it looks like the right thing to do and
it should noticeably simplify the sysfs locking.

Was this an oversight or is there some good reason we aren't using
i_mutex to lock the s_children list?

Eric



---

Subject: Re: [PATCH 06/14] sysfs: Rewrite sysfs_get_dentry
Posted by Tejun Heo on Wed, 01 Aug 2007 10:02:05 GMT
View Forum Message <> Reply to Message

Hello, Eric.

Eric W. Biederman wrote:
> I will look a little more and see. But right now it looks like the
> real problem with locking is that we use sysfs_mutex to lock the
> sysfs_dirent s_children list.

>
> Instead it really looks like we should use i_mutex from the appropriate
> inode.  Or is there a real performance problem with forcing the directory
> inodes in core when we modify the directories?

I don't think there is any performance problem.  Problems with using
i_mutex were...

* It was messy.  I don't remember all the details now but IIRC symlink
walk code was pretty complex.

* And more importantly, inodes are reclaimable and might or might not be
there.

--
tejun

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

---

Subject: Re: [PATCH 06/14] sysfs: Rewrite sysfs_get_dentry
Posted by ebiederm on Wed, 01 Aug 2007 17:07:39 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> Hello, Eric.
>
> Eric W. Biederman wrote:
>> I will look a little more and see.  But right now it looks like the
>> real problem with locking is that we use sysfs_mutex to lock the
>> sysfs_dirent s_children list.
>>
>> Instead it really looks like we should use i_mutex from the appropriate
>> inode.  Or is there a real performance problem with forcing the directory
>> inodes in core when we modify the directories?
>
> I don't think there is any performance problem.  Problems with using
> i_mutex were...
>
> * It was messy.  I don't remember all the details now but IIRC symlink
> walk code was pretty complex.
>
> * And more importantly, inodes are reclaimable and might or might not be
> there.

Yes.  But we can always force inodes into the cache when we need them.
When I complete it I will have to show you a patch using the inode lock
for locking directory modifications.  From what I can tell so far it allows
me to fix the weird lock order problems and generally simplify the locking.

Eric

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

---

Subject: Re: [PATCH 06/14] sysfs: Rewrite sysfs_get_dentry
Posted by Tejun Heo on Wed, 01 Aug 2007 17:20:38 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Tejun Heo <htejun@gmail.com> writes:
>
>> Hello, Eric.
>>
>> Eric W. Biederman wrote:
>>> I will look a little more and see.  But right now it looks like the
>>> real problem with locking is that we use sysfs_mutex to lock the
>>> sysfs_dirent s_children list.
>>>
>>> Instead it really looks like we should use i_mutex from the appropriate
>>> inode.  Or is there a real performance problem with forcing the directory
>>> inodes in core when we modify the directories?
>> I don't think there is any performance problem.  Problems with using
>> i_mutex were...
>>
>> * It was messy.  I don't remember all the details now but IIRC symlink
>> walk code was pretty complex.
>>
>> * And more importantly, inodes are reclaimable and might or might not be
>> there.
>
> Yes.  But we can always force inodes into the cache when we need them.
> When I complete it I will have to show you a patch using the inode lock
> for locking directory modifications.  From what I can tell so far it allows
> me to fix the weird lock order problems and generally simplify the locking.

Hmmm... I think we can live with a bit of complexity in
sysfs_get_dentry().  It's very well localized and not even long.  I have
been trying hard to untangle sysfs internals from vfs and have a bit of
resistance against going back.  But, then again, if we can achieve
something better and simpler, why not?

--
tejun

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

Subject: Re: [PATCH 06/14] sysfs: Rewrite sysfs_get_dentry
Posted by ebiederm on Wed, 01 Aug 2007 17:40:06 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> Hmmm... I think we can live with a bit of complexity in
> sysfs_get_dentry(). It's very well localized and not even long. I have
> been trying hard to untangle sysfs internals from vfs and have a bit of
> resistance against going back. But, then again, if we can achieve
> something better and simpler, why not?

Right now my real challenge is untangling sysfs_move_dir. In the
case of multiple mounts of the sysfs tree.

My practical problem is that I need to hold a lock for the sysfs
dirents and while that lock is held I need to call sysfs_get_dentry
for the destination directory once for each superblock.

It might be that some kind of reader-writer lock strategy is what
I need to untangle this mess. Rather then making changing to i_mutex.
All I know is at the moment it is taking a lot of code reading and
brain storm to come of with something that is easy to maintain.

Eric

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

Subject: Re: [PATCH 06/14] sysfs: Rewrite sysfs_get_dentry
Posted by Tejun Heo on Wed, 01 Aug 2007 17:51:19 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Tejun Heo <htejun@gmail.com> writes:
>

>> Hmmm... I think we can live with a bit of complexity in
>> sysfs_get_dentry(). It's very well localized and not even long. I have
>> been trying hard to untangle sysfs internals from vfs and have a bit of
>> resistance against going back. But, then again, if we can achieve
>> something better and simpler, why not?
>
> Right now my real challenge is untangling sysfs_move_dir. In the
> case of multiple mounts of the sysfs tree.

OIC. Yeah, I guess moving a sd with multiple dentries attached could be
pretty difficult.

> My practical problem is that I need to hold a lock for the sysfs
> dirents and while that lock is held I need to call sysfs_get_dentry
> for the destination directory once for each superblock.
>
> It might be that some kind of reader-writer lock strategy is what
> I need to untangle this mess. Rather then making changing to i_mutex.
> All I know is at the moment it is taking a lot of code reading and
> brain storm to come of with something that is easy to maintain.

Just in case, sysfs used to have sysfs_rename_rwsem to protect
move/rename against tree walking, which became unnecessary after i_mutex
-> sysfs_mutex conversion. Move/rename can use stupid big fat locks if
that helps.

Good luck.

--
tejun



---

Subject: Re: [PATCH 06/14] sysfs: Rewrite sysfs_get_dentry
Posted by Cornelia Huck on Thu, 02 Aug 2007 08:28:19 GMT
View Forum Message <> Reply to Message

On Thu, 02 Aug 2007 02:51:19 +0900,
Tejun Heo <htejun@gmail.com> wrote:

> Eric W. Biederman wrote:
> > My practical problem is that I need to hold a lock for the sysfs
> > dirents and while that lock is held I need to call sysfs_get_dentry
> > for the destination directory once for each superblock.
> >

> > It might be that some kind of reader-writer lock strategy is what
> > I need to untangle this mess.  Rather then making changing to i_mutex.
> > All I know is at the moment it is taking a lot of code reading and
> > brain storm to come of with something that is easy to maintain.
>
> Just in case, sysfs used to have sysfs_rename_rwsem to protect
> move/rename against tree walking, which became unnecessary after i_mutex
> -> sysfs_mutex conversion.  Move/rename can use stupid big fat locks if
> that helps.

I second that. Reintroduction of sysfs_rename_rwsem or something
similar may be the best way to avoid headaches.



---

Subject: Re: [PATCH 06/14] sysfs: Rewrite sysfs_get_dentry
Posted by ebiederm on Fri, 03 Aug 2007 19:29:19 GMT
View Forum Message <> Reply to Message

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

> On Thu, 02 Aug 2007 02:51:19 +0900,
> Tejun Heo <htejun@gmail.com> wrote:
>
>> Eric W. Biederman wrote:
>> > My practical problem is that I need to hold a lock for the sysfs
>> > dirents and while that lock is held I need to call sysfs_get_dentry
>> > for the destination directory once for each superblock.
>> >
>> > It might be that some kind of reader-writer lock strategy is what
>> > I need to untangle this mess.  Rather then making changing to i_mutex.
>> > All I know is at the moment it is taking a lot of code reading and
>> > brain storm to come of with something that is easy to maintain.
>>
>> Just in case, sysfs used to have sysfs_rename_rwsem to protect
>> move/rename against tree walking, which became unnecessary after i_mutex
>> -> sysfs_mutex conversion.  Move/rename can use stupid big fat locks if
>> that helps.
>
> I second that. Reintroduction of sysfs_rename_rwsem or something
> similar may be the best way to avoid headaches.

I guess I haven't looked at that because we already have a big fact
lock we just have an ordering problem in taking that lock.  Introducing
another lock just because of that doesn't quite feel right.

I currently have two practical solutions on the table.

- Make the s_sibling list walk RCU safe so it does not require us to
  grab the sysfs_mutex.  This works but is a bit complicated.

- Just kill all of the dentries in sysfs_move_dir and sysfs_rename_dir.
  The semantics are that no one can be using the device at the time
  of a rename or a move (as I read the callers) so dropping the dentries
  and making it look like a delete/add pair to the users should be
  acceptable and a whole lot simpler.

Eric

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

---

Subject: Re: [PATCH 06/14] sysfs: Rewrite sysfs_get_dentry
Posted by Cornelia Huck on Mon, 06 Aug 2007 15:06:36 GMT
View Forum Message <> Reply to Message

On Fri, 03 Aug 2007 13:29:19 -0600,
ebiederm@xmission.com (Eric W. Biederman) wrote:

> I currently have two practical solutions on the table.
>
> - Make the s_sibling list walk RCU safe so it does not require us to
>   grab the sysfs_mutex.  This works but is a bit complicated.

But it would be nice, I guess :)

>
> - Just kill all of the dentries in sysfs_move_dir and sysfs_rename_dir.
>   The semantics are that no one can be using the device at the time
>   of a rename or a move (as I read the callers) so dropping the dentries
>   and making it look like a delete/add pair to the users should be
>   acceptable and a whole lot simpler.

Hm, I'm not sure I understand you here. What do you mean by "using the
device"? Do you mean dropping the sysfs_dirents? Or am I just confused?

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

---