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;
```