Subject: [PATCH 02/15] sysfs: Support for preventing unmounts.
Posted by ebiederm on Fri, 04 Jul 2008 01:07:04 GMT
View Forum Message <> Reply to Message

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

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

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

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

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

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

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

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

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
Signed-off-by: Benjamin Thery <benjamin.thery@bull.net>
Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>
Acked-by: Tejun Heo <tj@kernel.org>
---
 fs/sysfs/mount.c | 79 ++-----
 fs/sysfs/sysfs.h | 10 +++++++
 2 files changed, 81 insertions(+), 8 deletions(-)

diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index 9f328d2..c812cc4 100644

```
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -41,47 +41,110 @@ struct sysfs_dirent sysfs_root = {

 static int sysfs_fill_super(struct super_block *sb, void *data, int silent)
 {
- struct inode *inode;
- struct dentry *root;
+ struct sysfs_super_info *info = NULL;
+ struct inode *inode = NULL;
+ struct dentry *root = NULL;
+ int error;

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

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

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

```
  return 0;
+
+out_err:
+ dput(root);
+ iput(inode);
+ kfree(info);
+ if (sysfs_sb == sb)
+  sysfs_sb = NULL;
+ return error;
 }

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

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

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

```
+  sysfs_info(sb)->grabbed = 1;
+ }
+ spin_unlock(&sb_lock);
+}
+
+void sysfs_release_supers(void)
+{
+ /* must hold sysfs_rename_mutex */
+ struct super_block *sb;
+restart:
+ spin_lock(&sb_lock);
+ list_for_each_entry(sb, &sysfs_fs_type.fs_supers, s_instances) {
+  if (!sysfs_info(sb)->grabbed)
+   continue;
+  sysfs_info(sb)->grabbed = 0;
+  spin_unlock(&sb_lock);
+  deactivate_super(sb);
+  goto restart;
+ }
+ spin_unlock(&sb_lock);
+}
+
 int __init sysfs_init(void)
 {
  int err = -ENOMEM;
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index 2915959..0fdc3de 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -85,6 +85,12 @@ struct sysfs_addrm_cxt {
  int   cnt;
 };

+struct sysfs_super_info {
+ int grabbed;
+};
+
+#define sysfs_info(SB) ((struct sysfs_super_info *)(SB)->s_fs_info)
+
 /*
  * mount.c
  */
@@ -92,6 +98,10 @@ extern struct sysfs_dirent sysfs_root;
 extern struct super_block *sysfs_sb;
 extern struct kmem_cache *sysfs_dir_cachep;
 extern struct vfsmount *sysfs_mount;
+extern struct file_system_type sysfs_fs_type;
+
```

```
+void sysfs_grab_supers(void);
+void sysfs_release_supers(void);

 /*
  * dir.c
--
1.5.3.rc6.17.g1911
```

_____

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