Subject: [PATCH 08/15] sysfs: Make sysfs_mount static once again.
Posted by ebiederm on Fri, 04 Jul 2008 01:14:36 GMT
View Forum Message <> Reply to Message

Accessing the internal sysfs_mount is error prone in the context
of multiple super blocks, and nothing needs it. Not even the
sysfs crash debugging patch (although it did in an earlier version).

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

diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index c812cc4..99974f0 100644
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -22,7 +22,7 @@
 /* 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/sysfs.h b/fs/sysfs/sysfs.h
index 5ee5d0a..33b3c73 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -97,7 +97,6 @@ struct sysfs_super_info {
 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);
--
1.5.3.rc6.17.g1911

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

Subject: [PATCH 09/15] sysfs: Implement sysfs tagged directory support.
Posted by ebiederm on Fri, 04 Jul 2008 01:16:08 GMT
View Forum Message <> Reply to Message

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

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

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

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

To achieve a reace free design all tagged directories are created and
managed by sysfs itself.

Users of this interface:
- define a type in the sysfs_tag_type enumeration.
- call sysfs_register_tag_types with the type and it's operations
- call sysfs_make_tagged_dir with the tag type on directories
  to be managed by this tag type
- sysfs_exit_tag when an individual tag is no longer valid

- Implement mount_tag() which returns the tag of the calling process
  so we can attach it to a sysfs superblock.
- Implement ktype.sysfs_tag() which returns the tag of a syfs kobject.

Everything else is left up to sysfs and the driver layer.

For the network namespace mount_tag and sysfs_tag are essentially
one line functions, and look to remain that.

Tags are currently represented a const void * pointers as that is
both generic, prevides enough information for equality comparisons,
and is trivial to create for current users, as it is just the
existing namespace pointer.

The work needed in sysfs is more extensive. At each directory
or symlink creating I need to check if the directory it is being
created in is a tagged directory and if so generate the appropriate
tag to place on the sysfs_dirent. Likewise at each symlink or
directory removal I need to check if the sysfs directory it is
being removed from is a tagged directory and if so figure out
which tag goes along with the name I am deleting.

Currently only directories which hold kobjects, and
symlinks are supported. There is not enough information
in the current file attribute interfaces to give us anything
to discriminate on which makes it useless, and there are
no potential users which makes it an uninteresting problem
to solve.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
Signed-off-by: Benjamin Thery <benjamin.thery@bull.net>
---
 fs/sysfs/bin.c | 2 +-
 fs/sysfs/dir.c | 139 ++-----
 fs/sysfs/file.c | 10 ++--
 fs/sysfs/group.c | 4 +-
 fs/sysfs/inode.c | 7 ++-
 fs/sysfs/mount.c | 115 ++++++++++++++++++++++++++++++++++++++--
 fs/sysfs/symlink.c | 2 +-
 fs/sysfs/sysfs.h | 19 ++++++-
 include/linux/kobject.h | 1 +
 include/linux/sysfs.h | 29 ++++++++++
 10 files changed, 295 insertions(+), 33 deletions(-)

diff --git a/fs/sysfs/bin.c b/fs/sysfs/bin.c
index 006fc64..86e1128 100644
--- a/fs/sysfs/bin.c
+++ b/fs/sysfs/bin.c
@@ -252,7 +252,7 @@ int sysfs_create_bin_file(struct kobject * kobj, struct bin_attribute * attr)

 void sysfs_remove_bin_file(struct kobject * kobj, struct bin_attribute * attr)
 {
- sysfs_hash_and_remove(kobj->sd, attr->attr.name);
+ sysfs_hash_and_remove(kobj, kobj->sd, attr->attr.name);
 }

 EXPORT_SYMBOL_GPL(sysfs_create_bin_file);
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index b2d92ea..6dc3376 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -30,6 +30,30 @@ DEFINE_SPINLOCK(sysfs_assoc_lock);

```
 static DEFINE_SPINLOCK(sysfs_ino_lock);
 static DEFINE_IDA(sysfs_ino_ida);

+static const void *sysfs_creation_tag(struct sysfs_dirent *parent_sd,
+          struct sysfs_dirent *sd)
+{
+ const void *tag = NULL;
+
+ if (sysfs_tag_type(parent_sd)) {
+  struct kobject *kobj;
+  switch (sysfs_type(sd)) {
+  case SYSFS_DIR:
+   kobj = sd->s_dir.kobj;
+   break;
+  case SYSFS_KOBJ_LINK:
+   kobj = sd->s_symlink.target_sd->s_dir.kobj;
+   break;
+  default:
+   BUG();
+  }
+  tag = kobj->ktype->sysfs_tag(kobj);
+  /* NULL tags are reserved for internal use */
+  BUG_ON(tag == NULL);
+ }
+ return tag;
+}
+
 /**
  * sysfs_link_sibling - link sysfs_dirent into sibling list
  * @sd: sysfs_dirent of interest
@@ -101,8 +125,19 @@ static void sysfs_unlink_sibling(struct sysfs_dirent *sd)
 struct dentry *sysfs_get_dentry(struct super_block *sb,
     struct sysfs_dirent *sd)
 {
- struct dentry *dentry = dget(sb->s_root);
+ struct dentry *dentry;
+
+ /* Bail if this sd won't show up in this superblock */
+ if (sd->s_parent) {
+  enum sysfs_tag_type type;
+  const void *tag;
+  type = sysfs_tag_type(sd->s_parent);
+  tag = sysfs_info(sb)->tag[type];
+  if (sd->s_tag != tag)
+   return ERR_PTR(-EXDEV);
+ }

+ dentry = dget(sb->s_root);
```

```
  while (dentry->d_fsdata != sd) {
   struct sysfs_dirent *cur;
   struct dentry *parent;
@@ -421,10 +456,15 @@ void sysfs_addrm_start(struct sysfs_addrm_cxt *acxt,
  */
 int sysfs_add_one(struct sysfs_addrm_cxt *acxt, struct sysfs_dirent *sd)
 {
- if (sysfs_find_dirent(acxt->parent_sd, sd->s_name))
+ const void *tag = NULL;
+
+ tag = sysfs_creation_tag(acxt->parent_sd, sd);
+
+ if (sysfs_find_dirent(acxt->parent_sd, tag, sd->s_name))
   return -EEXIST;

  sd->s_parent = sysfs_get(acxt->parent_sd);
+ sd->s_tag = tag;

  if (sysfs_type(sd) == SYSFS_DIR && acxt->parent_inode)
   inc_nlink(acxt->parent_inode);
@@ -572,13 +612,17 @@ void sysfs_addrm_finish(struct sysfs_addrm_cxt *acxt)
  * Pointer to sysfs_dirent if found, NULL if not.
  */
 struct sysfs_dirent *sysfs_find_dirent(struct sysfs_dirent *parent_sd,
+           const void *tag,
            const unsigned char *name)
 {
  struct sysfs_dirent *sd;

- for (sd = parent_sd->s_dir.children; sd; sd = sd->s_sibling)
+ for (sd = parent_sd->s_dir.children; sd; sd = sd->s_sibling) {
+  if (sd->s_tag != tag)
+   continue;
   if (!strcmp(sd->s_name, name))
    return sd;
+ }
  return NULL;
 }

@@ -602,7 +646,7 @@ struct sysfs_dirent *sysfs_get_dirent(struct sysfs_dirent *parent_sd,
  struct sysfs_dirent *sd;

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

```
@@ -668,13 +712,18 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry
*dentry,
     struct nameidata *nd)
 {
  struct dentry *ret = NULL;
- struct sysfs_dirent *parent_sd = dentry->d_parent->d_fsdata;
+ struct dentry *parent = dentry->d_parent;
+ struct sysfs_dirent *parent_sd = parent->d_fsdata;
  struct sysfs_dirent *sd;
  struct inode *inode;
+ enum sysfs_tag_type type;
+ const void *tag;

  mutex_lock(&sysfs_mutex);

- sd = sysfs_find_dirent(parent_sd, dentry->d_name.name);
+ type = sysfs_tag_type(parent_sd);
+ tag = sysfs_info(parent->d_sb)->tag[type];
+ sd = sysfs_find_dirent(parent_sd, tag, dentry->d_name.name);

  /* no such entry */
  if (!sd) {
@@ -882,19 +931,24 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  struct sysfs_rename_struct *srs;
  struct inode *parent_inode = NULL;
  const char *dup_name = NULL;
+ const void *old_tag, *tag;
  int error;

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

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

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

```
  error = -ENOMEM;
  mutex_lock(&sysfs_mutex);
@@ -907,7 +961,7 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  mutex_lock(&sysfs_mutex);

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

  /* rename sysfs_dirent */
@@ -918,6 +972,7 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)

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

  /* rename */
  list_for_each_entry(srs, &todo, list) {
@@ -925,6 +980,20 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
   d_move(srs->old_dentry, srs->new_dentry);
  }

+ /* If we are moving across superblocks drop the dcache entries */
+ if (old_tag != tag) {
+  struct super_block *sb;
+  struct dentry *dentry;
+  list_for_each_entry(sb, &sysfs_fs_type.fs_supers, s_instances) {
+   dentry = __sysfs_get_dentry(sb, sd);
+   if (!dentry)
+    continue;
+   shrink_dcache_parent(dentry);
+   d_drop(dentry);
+   dput(dentry);
+  }
+ }
+
  error = 0;
 out_unlock:
  mutex_unlock(&sysfs_mutex);
@@ -947,11 +1016,13 @@ int sysfs_move_dir(struct kobject *kobj, struct kobject
*new_parent_kobj)
  struct sysfs_rename_struct *srs;
  struct inode *old_parent_inode = NULL, *new_parent_inode = NULL;
  int error;
+ const void *tag;
```

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

  error = 0;
  if (sd->s_parent == new_parent_sd)
@@ -985,7 +1056,7 @@ again:
  mutex_lock(&sysfs_mutex);

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

  error = 0;
@@ -1024,10 +1095,12 @@ static inline unsigned char dt_type(struct sysfs_dirent *sd)

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

  if (filp->f_pos == 0) {
   ino = parent_sd->s_ino;
@@ -1045,6 +1118,9 @@ static int sysfs_readdir(struct file * filp, void * dirent, filldir_t filldir)
  if ((filp->f_pos > 1) && (filp->f_pos < INT_MAX)) {
   mutex_lock(&sysfs_mutex);

+  type = sysfs_tag_type(parent_sd);
+  tag = sysfs_info(parent->d_sb)->tag[type];
+
   /* Skip the dentries we have already reported */
   pos = parent_sd->s_dir.children;
   while (pos && (filp->f_pos > pos->s_ino))
@@ -1054,6 +1130,9 @@ static int sysfs_readdir(struct file * filp, void * dirent, filldir_t filldir)
    const char * name;
    int len;

+   if (pos->s_tag != tag)
+    continue;
```

```
+
    name = pos->s_name;
    len = strlen(name);
    filp->f_pos = ino = pos->s_ino;
@@ -1074,3 +1153,35 @@ const struct file_operations sysfs_dir_operations = {
  .read  = generic_read_dir,
  .readdir = sysfs_readdir,
 };
+
+/**
+ * sysfs_make_tagged_dir - Require tags of all the entries in a directory.
+ * @kobj: object whose children should be filtered by tags
+ *
+ * Once tagging has been enabled on a directory the contents
+ * of the directory become dependent upon context captured when
+ * sysfs was mounted.
+ */
+int sysfs_make_tagged_dir(struct kobject *kobj, enum sysfs_tag_type type)
+{
+ struct sysfs_dirent *sd;
+ int err;
+
+ err = -ENOENT;
+ sd = kobj->sd;
+
+ mutex_lock(&sysfs_mutex);
+ err = -EINVAL;
+ /* We can only enable tagging when we have a valid tag type
+  * on empty directories where taggint has not already been
+  * enabled.
+  */
+ if ((type > SYSFS_TAG_TYPE_NONE) && (type < SYSFS_TAG_TYPES) &&
+     tag_ops[type] && !sysfs_tag_type(sd) &&
+     (sysfs_type(sd) == SYSFS_DIR) && !sd->s_dir.children) {
+  err = 0;
+  sd->s_flags |= (type << SYSFS_TAG_TYPE_SHIFT);
+ }
+ mutex_unlock(&sysfs_mutex);
+ return err;
+}
diff --git a/fs/sysfs/file.c b/fs/sysfs/file.c
index 5955ae9..be95fa2 100644
--- a/fs/sysfs/file.c
+++ b/fs/sysfs/file.c
@@ -461,9 +461,11 @@ void sysfs_notify(struct kobject *k, char *dir, char *attr)
  mutex_lock(&sysfs_mutex);

  if (sd && dir)
```

```
-  sd = sysfs_find_dirent(sd, dir);
+  /* only directories are tagged, so no need to pass
+   a tag explicitly */
+  sd = sysfs_find_dirent(sd, NULL, dir);
  if (sd && attr)
-  sd = sysfs_find_dirent(sd, attr);
+  sd = sysfs_find_dirent(sd, NULL, attr);
  if (sd) {
   struct sysfs_open_dirent *od;

@@ -636,7 +638,7 @@ EXPORT_SYMBOL_GPL(sysfs_chmod_file);

 void sysfs_remove_file(struct kobject * kobj, const struct attribute * attr)
 {
- sysfs_hash_and_remove(kobj->sd, attr->name);
+ sysfs_hash_and_remove(kobj, kobj->sd, attr->name);
 }


@@ -656,7 +658,7 @@ void sysfs_remove_file_from_group(struct kobject *kobj,
  else
   dir_sd = sysfs_get(kobj->sd);
  if (dir_sd) {
-  sysfs_hash_and_remove(dir_sd, attr->name);
+  sysfs_hash_and_remove(kobj, dir_sd, attr->name);
   sysfs_put(dir_sd);
  }
 }
diff --git a/fs/sysfs/group.c b/fs/sysfs/group.c
index eeba384..b6693b4 100644
--- a/fs/sysfs/group.c
+++ b/fs/sysfs/group.c
@@ -23,7 +23,7 @@ static void remove_files(struct sysfs_dirent *dir_sd, struct kobject *kobj,
  int i;

  for (i = 0, attr = grp->attrs; *attr; i++, attr++)
-  sysfs_hash_and_remove(dir_sd, (*attr)->name);
+  sysfs_hash_and_remove(kobj, dir_sd, (*attr)->name);
 }

 static int create_files(struct sysfs_dirent *dir_sd, struct kobject *kobj,
@@ -39,7 +39,7 @@ static int create_files(struct sysfs_dirent *dir_sd, struct kobject *kobj,
    * visibility.  Do this by first removing then
    * re-adding (if required) the file */
   if (update)
-   sysfs_hash_and_remove(dir_sd, (*attr)->name);
+   sysfs_hash_and_remove(kobj, dir_sd, (*attr)->name);
   if (grp->is_visible) {
```

```
    mode = grp->is_visible(kobj, *attr, i);
    if (!mode)
diff --git a/fs/sysfs/inode.c b/fs/sysfs/inode.c
index 80f8fd4..b5fc78a 100644
--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -226,17 +226,20 @@ struct inode * sysfs_get_inode(struct sysfs_dirent *sd)
  return inode;
 }

-int sysfs_hash_and_remove(struct sysfs_dirent *dir_sd, const char *name)
+int sysfs_hash_and_remove(struct kobject *kobj, struct sysfs_dirent *dir_sd,
+      const char *name)
 {
  struct sysfs_addrm_cxt acxt;
  struct sysfs_dirent *sd;
+ const void *tag;

  if (!dir_sd)
   return -ENOENT;

  sysfs_addrm_start(&acxt, dir_sd);
+ tag = kobj->sd->s_tag;

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

diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index 99974f0..c4a3022 100644
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -34,12 +34,15 @@ static const struct super_operations sysfs_ops = {
 struct sysfs_dirent sysfs_root = {
  .s_name  = "",
  .s_count = ATOMIC_INIT(1),
- .s_flags = SYSFS_DIR,
+ .s_flags = SYSFS_DIR | (SYSFS_TAG_TYPE_NONE << SYSFS_TAG_TYPE_SHIFT),
  .s_mode  = S_IFDIR | S_IRWXU | S_IRUGO | S_IXUGO,
  .s_ino  = 1,
 };

-static int sysfs_fill_super(struct super_block *sb, void *data, int silent)
+struct sysfs_tag_type_operations *tag_ops[SYSFS_TAG_TYPES];
+
+static int sysfs_fill_super(struct super_block *sb, void *data, int silent,
+ const void *tags[SYSFS_TAG_TYPES])
```

```
 {
  struct sysfs_super_info *info = NULL;
  struct inode *inode = NULL;
@@ -75,8 +78,10 @@ static int sysfs_fill_super(struct super_block *sb, void *data, int silent)
   goto out_err;
  }
  root->d_fsdata = &sysfs_root;
+ root->d_sb = sb;
  sb->s_root = root;
  sb->s_fs_info = info;
+ memcpy(info->tag, tags, sizeof(info->tag[0])*SYSFS_TAG_TYPES);
  return 0;

 out_err:
@@ -88,20 +93,74 @@ out_err:
  return error;
 }

+static int sysfs_test_super(struct super_block *sb, void *ptr)
+{
+ const void **tag = ptr;
+ struct sysfs_super_info *info = sysfs_info(sb);
+ enum sysfs_tag_type type;
+ int found = 1;
+
+ for (type = SYSFS_TAG_TYPE_NONE; type < SYSFS_TAG_TYPES; type++) {
+  if (info->tag[type] != tag[type]) {
+   found = 0;
+   break;
+  }
+ }
+
+ return found;
+}
+
 static int sysfs_get_sb(struct file_system_type *fs_type,
  int flags, const char *dev_name, void *data, struct vfsmount *mnt)
 {
- int rc;
+ const void *tag[SYSFS_TAG_TYPES];
+ struct super_block *sb;
+ int error;
+ enum sysfs_tag_type type;
+
+ for (type = SYSFS_TAG_TYPE_NONE; type < SYSFS_TAG_TYPES; type++) {
+  tag[type] = NULL;
+  if (!tag_ops[type])
+   continue;
```

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

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

 void sysfs_grab_supers(void)
@@ -145,6 +204,50 @@ restart:
  spin_unlock(&sb_lock);
 }
```

```
+int sysfs_register_tag_type(enum sysfs_tag_type type, struct sysfs_tag_type_operations *ops)
+{
+ int error;
+
+ mutex_lock(&sysfs_rename_mutex);
+
+ error = -EINVAL;
+ if (type >= SYSFS_TAG_TYPES)
+  goto out;
+
+ error = -EINVAL;
+ if (type <= SYSFS_TAG_TYPE_NONE)
+  goto out;
+
+ error = -EBUSY;
+ if (tag_ops[type])
+  goto out;
+
+ error = 0;
+ tag_ops[type] = ops;
+
+out:
+ mutex_unlock(&sysfs_rename_mutex);
+ return error;
+}
+
+void sysfs_exit_tag(enum sysfs_tag_type type, const void *tag)
+{
+ /* Allow the tag to go away while sysfs is still mounted. */
+ struct super_block *sb;
+ mutex_lock(&sysfs_rename_mutex);
+ sysfs_grab_supers();
+ mutex_lock(&sysfs_mutex);
+ list_for_each_entry(sb, &sysfs_fs_type.fs_supers, s_instances) {
+  struct sysfs_super_info *info = sysfs_info(sb);
+  if (info->tag[type] != tag)
+   continue;
+  info->tag[type] = NULL;
+ }
+ mutex_unlock(&sysfs_mutex);
+ sysfs_release_supers();
+ mutex_unlock(&sysfs_rename_mutex);
+}
+
 int __init sysfs_init(void)
 {
  int err = -ENOMEM;
```

```
diff --git a/fs/sysfs/symlink.c b/fs/sysfs/symlink.c
index 817f596..de9a5c0 100644
--- a/fs/sysfs/symlink.c
+++ b/fs/sysfs/symlink.c
@@ -94,7 +94,7 @@ void sysfs_remove_link(struct kobject * kobj, const char * name)
  else
   parent_sd = kobj->sd;

- sysfs_hash_and_remove(parent_sd, name);
+ sysfs_hash_and_remove(kobj, parent_sd, name);
 }

 static int sysfs_get_target_path(struct sysfs_dirent *parent_sd,
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index 33b3c73..4128e6f 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -45,6 +45,7 @@ struct sysfs_dirent {
  struct sysfs_dirent *s_sibling;
  const char  *s_name;

+ const void  *s_tag;
  union {
   struct sysfs_elem_dir  s_dir;
   struct sysfs_elem_symlink s_symlink;
@@ -67,14 +68,22 @@ struct sysfs_dirent {
 #define SYSFS_KOBJ_LINK   0x0008
 #define SYSFS_COPY_NAME   (SYSFS_DIR | SYSFS_KOBJ_LINK)

-#define SYSFS_FLAG_MASK   ~SYSFS_TYPE_MASK
-#define SYSFS_FLAG_REMOVED  0x0200
+#define SYSFS_TAG_TYPE_MASK  0xff00
+#define SYSFS_TAG_TYPE_SHIFT  8
+
+#define SYSFS_FLAG_MASK   ~(SYSFS_TYPE_MASK | SYSFS_TAG_TYPE_MASK)
+#define SYSFS_FLAG_REMOVED  0x020000

 static inline unsigned int sysfs_type(struct sysfs_dirent *sd)
 {
  return sd->s_flags & SYSFS_TYPE_MASK;
 }

+static inline enum sysfs_tag_type sysfs_tag_type(struct sysfs_dirent *sd)
+{
+ return (sd->s_flags & SYSFS_TAG_TYPE_MASK) >> SYSFS_TAG_TYPE_SHIFT;
+}
+
 /*
```

```
 * Context structure to be used while adding/removing nodes.
 */
@@ -87,6 +96,7 @@ struct sysfs_addrm_cxt {

 struct sysfs_super_info {
  int grabbed;
+ const void *tag[SYSFS_TAG_TYPES];
 };

 #define sysfs_info(SB) ((struct sysfs_super_info *)(SB)->s_fs_info)
@@ -98,6 +108,7 @@ extern struct sysfs_dirent sysfs_root;
 extern struct super_block *sysfs_sb;
 extern struct kmem_cache *sysfs_dir_cachep;
 extern struct file_system_type sysfs_fs_type;
+extern struct sysfs_tag_type_operations *tag_ops[SYSFS_TAG_TYPES];

 void sysfs_grab_supers(void);
 void sysfs_release_supers(void);
@@ -123,6 +134,7 @@ void sysfs_remove_one(struct sysfs_addrm_cxt *acxt, struct sysfs_dirent
*sd);
 void sysfs_addrm_finish(struct sysfs_addrm_cxt *acxt);

 struct sysfs_dirent *sysfs_find_dirent(struct sysfs_dirent *parent_sd,
+           const void *tag,
            const unsigned char *name);
 struct sysfs_dirent *sysfs_get_dirent(struct sysfs_dirent *parent_sd,
           const unsigned char *name);
@@ -155,7 +167,8 @@ static inline void sysfs_put(struct sysfs_dirent *sd)
 struct inode *sysfs_get_inode(struct sysfs_dirent *sd);
 int sysfs_sd_setattr(struct sysfs_dirent *sd, struct inode *inode, struct iattr *iattr);
 int sysfs_setattr(struct dentry *dentry, struct iattr *iattr);
-int sysfs_hash_and_remove(struct sysfs_dirent *dir_sd, const char *name);
+int sysfs_hash_and_remove(struct kobject *kobj, struct sysfs_dirent *dir_sd,
+     const char *name);
 int sysfs_inode_init(void);

 /*
diff --git a/include/linux/kobject.h b/include/linux/kobject.h
index 60f0d41..6bc38f8 100644
--- a/include/linux/kobject.h
+++ b/include/linux/kobject.h
@@ -105,6 +105,7 @@ struct kobj_type {
  void (*release)(struct kobject *kobj);
  struct sysfs_ops *sysfs_ops;
  struct attribute **default_attrs;
+ const void *(*sysfs_tag)(struct kobject *kobj);
 };
```

```
 struct kobj_uevent_env {
diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
index f7e43ed..8fa97f0 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -78,6 +78,15 @@ struct sysfs_ops {
  ssize_t (*store)(struct kobject *,struct attribute *,const char *, size_t);
 };

+enum sysfs_tag_type {
+ SYSFS_TAG_TYPE_NONE = 0,
+ SYSFS_TAG_TYPES
+};
+
+struct sysfs_tag_type_operations {
+ const void *(*mount_tag)(void);
+};
+
 #ifdef CONFIG_SYSFS

 int sysfs_schedule_callback(struct kobject *kobj, void (*func)(void *),
@@ -117,6 +126,10 @@ void sysfs_remove_file_from_group(struct kobject *kobj,
 void sysfs_notify(struct kobject *kobj, char *dir, char *attr);
 void sysfs_printk_last_file(void);

+int sysfs_make_tagged_dir(struct kobject *, enum sysfs_tag_type tag_type);
+int sysfs_register_tag_type(enum sysfs_tag_type type, struct sysfs_tag_type_operations *ops);
+void sysfs_exit_tag(enum sysfs_tag_type type, const void *tag);
+
 extern int __must_check sysfs_init(void);

 #else /* CONFIG_SYSFS */
@@ -217,6 +230,22 @@ static inline void sysfs_notify(struct kobject *kobj, char *dir, char *attr)
 {
 }

+staticn inline int sysfs_make_tagged_dir(struct kobject *kobj,
+      enum sysfs_tag_type tag_type)
+{
+ return 0;
+}
+
+static inline int sysfs_register_tag_type(enum sysfs_tag_type type,
+     struct sysfs_tag_type_operations *ops)
+{
+ return 0;
+}
+
```

```
+static inline void sysfs_exit_tag(enum sysfs_tag_type type, const void *tag)
+{
+}
+
 static inline int __must_check sysfs_init(void)
 {
  return 0;
--
1.5.3.rc6.17.g1911
```

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

---

Subject: Re: [PATCH 08/15] sysfs: Make sysfs_mount static once again.
Posted by Tejun Heo on Fri, 04 Jul 2008 06:44:59 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Accessing the internal sysfs_mount is error prone in the context
> of multiple super blocks, and nothing needs it. Not even the
> sysfs crash debugging patch (although it did in an earlier version).
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

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

--
tejun

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

---