Subject: [PATCH 3/5] sysfs: Implement sysfs manged shadow directory support.
Posted by ebiederm on Fri, 06 Apr 2007 16:50:32 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 I want is for each network namespace to have it's own separate
set of directories. /sys/class/net/, /sys/devices/virtual/net,
and /sys/ ... /net/, and in each set I want to name them
/sys/class/net/, sys/devices/virtual/net/ and /sys/ ... /net/ respectively.

I looked and the VFS actually allows that. All that is needed is
for /sys/class/net to implement a follow link method to redirect
lookups to the real directory you want.

I am calling the concept of multiple directories all at the same path
in the filesystem shadow directories, the directory entry that
implements the follow_link method the shadow master, and the directories
that are the target of the follow link method shadow directories.

It turns out that just implementing a follow_link method is not
quite enough. The existince of directories of the form /sys/ ... /net/
can depend on the presence or absence of hotplug hardware, which
means I need a simple race free way to create and destroy these
directories.

To achieve a race free design all shadow directories are created
and managed by sysfs itself. The upper level code that knows what
shadow directories we need provides just two methods that enable
this:
  current_tag() - that returns a "void *" tag that identifies the context of
 the current process.
  kobject_tag(kobj) - that returns a "void *" tag that identifies the context
 a kobject should be in.
Everything else is left up to sysfs.

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

The work needed in sysfs is more extensive. At each directory or
symlink creation I need to check if the shadow directory it belongs
in exists and if it does not create it. Likewise at each symlink
or directory removal I need to check if sysfs directory it is being
removed from is a shadow directory and if this is the last object
in the shadow directory and if so to remove the shadow directory

as well.

I also need a bunch of boiler plate that properly finds, creates, and
removes/frees the shadow directories.

Doing all of that in sysfs isn't bad it is just a bit tedious. Being race
free is just a manner of ensure we have the directory inode mutex when
we add or remove a shadow directory. Trying to do this race free
anywhere besides in sysfs is very nasty, and requires unhealthy
amounts of information about how sysfs is implemented.

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
not potential users which makes it an uniteresting problem
to solve.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/bin.c | 2 +-
 fs/sysfs/dir.c | 370 ++---
 fs/sysfs/file.c | 4 +-
 fs/sysfs/group.c | 12 +-
 fs/sysfs/inode.c | 18 ++-
 fs/sysfs/symlink.c | 11 +-
 fs/sysfs/sysfs.h | 18 +++-
 include/linux/sysfs.h | 14 ++
 8 files changed, 409 insertions(+), 40 deletions(-)

diff --git a/fs/sysfs/bin.c b/fs/sysfs/bin.c
index d3b9f5f..a79ed4c 100644
--- a/fs/sysfs/bin.c
+++ b/fs/sysfs/bin.c
@@ -195,7 +195,7 @@ int sysfs_create_bin_file(struct kobject * kobj, struct bin_attribute * attr)

 void sysfs_remove_bin_file(struct kobject * kobj, struct bin_attribute * attr)
 {
- if (sysfs_hash_and_remove(kobj->dentry, attr->attr.name) < 0) {
+ if (sysfs_hash_and_remove(kobj, kobj->dentry, attr->attr.name) < 0) {
   printk(KERN_ERR "%s: "
    "bad dentry or inode or no such file: \"%s\"\n",
    __FUNCTION__, attr->attr.name);
diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index d6ab015..0802a73 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -123,21 +123,26 @@ static int init_symlink(struct inode * inode)

```
 static int create_dir(struct kobject * k, struct dentry * p,
        const char * n, struct dentry ** d)
 {
+ struct dentry *parent;
  int error;
  umode_t mode = S_IFDIR| S_IRWXU | S_IRUGO | S_IXUGO;

  mutex_lock(&p->d_inode->i_mutex);
- *d = lookup_one_len(n, p, strlen(n));
+ parent = sysfs_resolve_for_create(k, p);
+ if (IS_ERR(parent))
+  *d = parent;
+ else
+  *d = lookup_one_len(n, parent, strlen(n));
  if (!IS_ERR(*d)) {
-   if (sysfs_dirent_exist(p->d_fsdata, n))
+  if (sysfs_dirent_exist(parent->d_fsdata, n))
      error = -EEXIST;
     else
-   error = sysfs_make_dirent(p->d_fsdata, *d, k, mode,
+   error = sysfs_make_dirent(parent->d_fsdata, *d, k, mode,
        SYSFS_DIR);
   if (!error) {
    error = sysfs_create(*d, mode, init_dir);
    if (!error) {
-    inc_nlink(p->d_inode);
+    inc_nlink(parent->d_inode);
     (*d)->d_op = &sysfs_dentry_ops;
     d_rehash(*d);
    }
@@ -288,6 +293,7 @@ static void remove_dir(struct dentry * d)
    atomic_read(&d->d_count));

  mutex_unlock(&parent->d_inode->i_mutex);
+ sysfs_prune_shadow(parent);
  dput(parent);
 }

@@ -296,6 +302,55 @@ void sysfs_remove_subdir(struct dentry * d)
  remove_dir(d);
 }

+static void sysfs_empty_dir(struct dentry *dentry)
+{
+ struct sysfs_dirent *parent_sd, *sd, *tmp;
+
+ parent_sd = dentry->d_fsdata;
+ list_for_each_entry_safe(sd, tmp, &parent_sd->s_children, s_sibling) {
```

```
+  if (!sd->s_element || !(sd->s_type & SYSFS_NOT_PINNED))
+   continue;
+  list_del_init(&sd->s_sibling);
+  sysfs_drop_dentry(sd, dentry);
+  sysfs_put(sd);
+ }
+}
+
+static void __sysfs_remove_empty_shadow(struct dentry *shadow)
+{
+ struct sysfs_dirent *sd;
+
+ if (d_unhashed(shadow))
+  return;
+
+ sd = shadow->d_fsdata;
+ d_delete(shadow);
+ list_del_init(&sd->s_sibling);
+ simple_rmdir(shadow->d_inode, shadow);
+ sysfs_put(sd);
+}
+
+static void sysfs_remove_empty_shadow(struct dentry *shadow)
+{
+ mutex_lock(&shadow->d_inode->i_mutex);
+ __sysfs_remove_empty_shadow(shadow);
+ mutex_unlock(&shadow->d_inode->i_mutex);
+}
+
+static void sysfs_remove_shadows(struct dentry *dentry)
+{
+ struct sysfs_dirent *parent_sd, *sd, *tmp;
+
+ parent_sd = dentry->d_fsdata;
+ list_for_each_entry_safe(sd, tmp, &parent_sd->s_children, s_sibling) {
+  struct dentry *shadow;
+
+  shadow = dget(sd->s_dentry);
+  sysfs_empty_dir(shadow);
+  __sysfs_remove_empty_shadow(shadow);
+  dput(shadow);
+ }
+}

 /**
  * sysfs_remove_dir - remove an object's directory.
@@ -309,22 +364,16 @@ void sysfs_remove_subdir(struct dentry * d)
 void sysfs_remove_dir(struct kobject * kobj)
```

```
 {
  struct dentry * dentry = dget(kobj->dentry);
- struct sysfs_dirent * parent_sd;
- struct sysfs_dirent * sd, * tmp;

  if (!dentry)
   return;

  pr_debug("sysfs %s: removing dir\n",dentry->d_name.name);
  mutex_lock(&dentry->d_inode->i_mutex);
- parent_sd = dentry->d_fsdata;
- list_for_each_entry_safe(sd, tmp, &parent_sd->s_children, s_sibling) {
-  if (!sd->s_element || !(sd->s_type & SYSFS_NOT_PINNED))
-   continue;
-  list_del_init(&sd->s_sibling);
-  sysfs_drop_dentry(sd, dentry);
-  sysfs_put(sd);
- }
+ if (dentry->d_inode->i_op == &sysfs_dir_inode_operations)
+  sysfs_empty_dir(dentry);
+ else
+  sysfs_remove_shadows(dentry);
  mutex_unlock(&dentry->d_inode->i_mutex);

  remove_dir(dentry);
@@ -350,7 +399,7 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  down_write(&sysfs_rename_sem);
  mutex_lock(&inode->i_mutex);

- new_parent = kobj->dentry->d_parent;
+ new_parent = sysfs_resolve_for_create(kobj, kobj->dentry->d_parent);
  new_dentry = lookup_one_len(new_name, new_parent, strlen(new_name));
  if (!IS_ERR(new_dentry)) {
   if (new_dentry == kobj->dentry)
@@ -378,6 +427,7 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  mutex_unlock(&inode->i_mutex);
  up_write(&sysfs_rename_sem);

+ sysfs_prune_shadow(old_parent);
  dput(old_parent);

  return error;
@@ -385,24 +435,35 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)

 int sysfs_move_dir(struct kobject *kobj, struct kobject *new_parent)
 {
+ struct inode *old_parent_inode, *new_parent_inode;
  struct dentry *old_parent_dentry, *new_parent_dentry, *new_dentry;
```

```
  struct sysfs_dirent *new_parent_sd, *sd;
  int error;

- old_parent_dentry = kobj->parent ?
-  kobj->parent->dentry : sysfs_mount->mnt_sb->s_root;
+ old_parent_dentry = kobj->dentry->d_parent;
  new_parent_dentry = new_parent ?
   new_parent->dentry : sysfs_mount->mnt_sb->s_root;

- if (old_parent_dentry->d_inode == new_parent_dentry->d_inode)
+ old_parent_inode = old_parent_dentry->d_inode;
+ new_parent_inode = new_parent_dentry->d_inode;
+
+ if (old_parent_inode == new_parent_inode)
   return 0; /* nothing to move */
+
+ dget(old_parent_dentry);
 again:
- mutex_lock(&old_parent_dentry->d_inode->i_mutex);
- if (!mutex_trylock(&new_parent_dentry->d_inode->i_mutex)) {
-  mutex_unlock(&old_parent_dentry->d_inode->i_mutex);
+ mutex_lock(&old_parent_inode->i_mutex);
+ if (!mutex_trylock(&new_parent_inode->i_mutex)) {
+  mutex_unlock(&old_parent_inode->i_mutex);
   goto again;
  }

+ new_parent_dentry = sysfs_resolve_for_create(kobj, new_parent_dentry);
+ if (IS_ERR(new_parent_dentry)) {
+  error = PTR_ERR(new_parent_dentry);
+  goto out;
+ }
+
  new_parent_sd = new_parent_dentry->d_fsdata;
  sd = kobj->dentry->d_fsdata;

@@ -422,8 +483,11 @@ again:
  list_add(&sd->s_sibling, &new_parent_sd->s_children);

 out:
- mutex_unlock(&new_parent_dentry->d_inode->i_mutex);
- mutex_unlock(&old_parent_dentry->d_inode->i_mutex);
+ mutex_unlock(&new_parent_inode->i_mutex);
+ mutex_unlock(&old_parent_inode->i_mutex);
+
+ sysfs_prune_shadow(old_parent_dentry);
+ dput(old_parent_dentry);
```

```
  return error;
 }
@@ -486,6 +550,11 @@ static int sysfs_readdir(struct file * filp, void * dirent, filldir_t filldir)
     i++;
     /* fallthrough */
   default:
+   /* If I am the shadow master return nothing. */
+   if ((parent_sd->s_type == SYSFS_DIR) &&
+       (dentry->d_inode->i_op != &sysfs_dir_inode_operations))
+    return 0;
+
     if (filp->f_pos == 2)
      list_move(q, &parent_sd->s_children);

@@ -565,3 +634,258 @@ const struct file_operations sysfs_dir_operations = {
  .read  = generic_read_dir,
  .readdir = sysfs_readdir,
 };
+
+static struct dentry *find_shadow_dir(struct dentry *parent, void *target)
+{
+ /* Find the shadow directory for the specified tag */
+ struct sysfs_dirent *parent_sd, *sd;
+ struct dentry *result;
+
+ result = NULL;
+ parent_sd = parent->d_fsdata;
+ list_for_each_entry(sd, &parent_sd->s_children, s_sibling) {
+  struct sysfs_shadow_dir *sdd;
+  sdd = sd->s_element;
+  if (sdd->tag != target)
+   continue;
+  result = sd->s_dentry;
+  break;
+ }
+ return result;
+}
+
+static void *find_shadow_tag(struct kobject *kobj)
+{
+ /* Find the tag the current kobj is cached with */
+ struct sysfs_shadow_dir *sdd;
+ struct dentry *dir;
+ struct sysfs_dirent *parent_sd, *sd;
+ void *result;
+
+ result = ERR_PTR(-ENOENT);
+ dir = kobj->dentry;
```

```
+ sd = dir->d_parent->d_fsdata;
+ if (sd->s_type != SYSFS_SHADOW)
+  goto out;
+ sdd = sd->s_element;
+ parent_sd = sdd->kobj->dentry->d_fsdata;
+
+ list_for_each_entry(sd, &parent_sd->s_children, s_sibling) {
+  struct sysfs_shadow_dir *sdd;
+  struct dentry *dentry;
+  int found;
+  sdd = sd->s_element;
+
+  dentry = lookup_one_len(dir->d_name.name, sd->s_dentry,
+     dir->d_name.len);
+  found = (dentry == dir);
+  dput(dentry);
+
+  if (!found)
+   continue;
+
+  result = sdd->tag;
+  break;
+ }
+out:
+ return result;
+}
+
+static struct dentry *add_shadow_dir(struct dentry *dir, void *tag)
+{
+ struct sysfs_dirent *parent_sd;
+ struct sysfs_shadow_dir *sdd;
+ struct dentry *shadow;
+ struct inode *inode;
+ int err;
+
+ parent_sd = dir->d_fsdata;
+ inode = dir->d_inode;
+
+ err = -ENOMEM;
+ shadow = d_alloc(dir->d_parent, &dir->d_name);
+ if (!shadow)
+  goto error;
+
+ sdd = kmalloc(sizeof(*sdd), GFP_KERNEL);
+ if (!sdd)
+  goto error;
+ sdd->tag = tag;
+ sdd->kobj = parent_sd->s_element;
```

```
+
+ err = sysfs_make_dirent(parent_sd, shadow, sdd, inode->i_mode,
+    SYSFS_SHADOW);
+ if (err)
+  goto error;
+
+ d_instantiate(shadow, igrab(inode));
+
+ /* Since the shadow directory is reachable make it look
+  * like it is actually hashed.
+  */
+ shadow->d_hash.pprev = &shadow->d_hash.next;
+ shadow->d_hash.next = NULL;
+ shadow->d_flags &= ~DCACHE_UNHASHED;
+
+ inc_nlink(inode);
+ inc_nlink(dir->d_parent->d_inode);
+ shadow->d_op = &sysfs_dentry_ops;
+
+out:
+ return shadow;
+error:
+ dput(shadow);
+ shadow = ERR_PTR(err);
+ goto out;
+}
+
+void sysfs_prune_shadow(struct dentry *shadow)
+{
+ struct sysfs_dirent *sd;
+
+ if (!shadow)
+  return;
+ sd = shadow->d_fsdata;
+ if (sd->s_type != SYSFS_SHADOW)
+  return;
+
+ if (!list_empty(&sd->s_children))
+  return;
+
+ sysfs_remove_empty_shadow(shadow);
+}
+
+struct dentry *sysfs_resolve_for_create(struct kobject *kobj, struct dentry *dentry)
+{
+ const struct shadow_dir_operations *shadow_ops;
+ struct dentry *dir;
+ struct inode *inode;
```

```
+
+ inode = dentry->d_inode;
+ shadow_ops = inode->i_private;
+ if (!shadow_ops)
+  dir = dentry;
+ else {
+  struct sysfs_dirent *sd;
+  void *tag;
+  sd = dentry->d_fsdata;
+  if (sd->s_type == SYSFS_SHADOW) {
+   struct sysfs_shadow_dir *sdd = sd->s_element;
+   dentry = sdd->kobj->dentry;
+  }
+  tag = shadow_ops->kobject_tag(kobj);
+  dir = find_shadow_dir(dentry, tag);
+  if (!dir)
+   dir = add_shadow_dir(dentry, tag);
+ }
+ return dir;
+}
+
+struct dentry *sysfs_resolve_for_remove(struct kobject *kobj, struct dentry *dentry)
+{
+ const struct shadow_dir_operations *shadow_ops;
+ struct dentry *dir;
+ struct inode *inode;
+
+ inode = dentry->d_inode;
+ shadow_ops = inode->i_private;
+ if (!shadow_ops)
+  dir = dentry;
+ else {
+  struct sysfs_dirent *sd;
+  void *tag;
+  sd = dentry->d_fsdata;
+  if (sd->s_type == SYSFS_SHADOW) {
+   struct sysfs_shadow_dir *sdd = sd->s_element;
+   dentry = sdd->kobj->dentry;
+  }
+  tag = find_shadow_tag(kobj);
+  if (IS_ERR(tag))
+   dir = tag;
+  else
+   dir = find_shadow_dir(dentry, tag);
+  if (!dir)
+   dir = ERR_PTR(-ENOENT);
+ }
+ return dir;
```

```
+}
+
+static void *sysfs_shadow_follow_link(struct dentry *dentry, struct nameidata *nd)
+{
+ struct sysfs_dirent *sd;
+ struct dentry *dest;
+
+ sd = dentry->d_fsdata;
+ dest = NULL;
+ if (sd->s_type == SYSFS_DIR) {
+  const struct shadow_dir_operations *shadow_ops;
+  struct inode *inode;
+  inode = dentry->d_inode;
+  shadow_ops = inode->i_private;
+  mutex_lock(&inode->i_mutex);
+  dest = find_shadow_dir(dentry, shadow_ops->current_tag());
+  dget(dest);
+  mutex_unlock(&inode->i_mutex);
+ }
+ if (!dest)
+  dest = dget(dentry);
+ dput(nd->dentry);
+ nd->dentry = dest;
+
+ return NULL;
+}
+
+static struct inode_operations sysfs_shadow_inode_operations = {
+ .lookup  = sysfs_lookup,
+ .setattr = sysfs_setattr,
+ .follow_link = sysfs_shadow_follow_link,
+};
+
+
+/**
+ * sysfs_enable_shadowing - Automatically create shadows of a directory
+ * @kobj: object to automatically shadow
+ *
+ * Once shadowing has been enabled on a directory the contents
+ * of the directory become dependent upon context.
+ *
+ * shadow_ops->current_tag() returns the context for the current
+ * process.
+ *
+ * shadow_ops->kobject_tag() returns the context that a given kobj
+ * resides in.
+ *
+ * Using those methods the sysfs code on shadowed directories
```

```
+ * carefully stores the files so that when we lookup files
+ * we get the proper answer for our context.
+ *
+ * If the context of a kobject is changed it is expected that
+ * the kobject will be renamed so the appopriate sysfs data structures
+ * can be updated.
+ */
+int sysfs_enable_shadowing(struct kobject *kobj,
+ const struct shadow_dir_operations *shadow_ops)
+{
+ struct sysfs_dirent *sd;
+ struct dentry *dentry;
+ struct inode *inode;
+ int err;
+
+ dentry = kobj->dentry;
+ inode = dentry->d_inode;
+
+ mutex_lock(&inode->i_mutex);
+ /* We can only enable shadowing on empty directories
+  * where shadowing is not already enabled.
+  */
+ sd = dentry->d_fsdata;
+ err = -EINVAL;
+ if (!inode->i_private && list_empty(&sd->s_children)) {
+  inode->i_private = (void *)shadow_ops;
+  inode->i_op = &sysfs_shadow_inode_operations;
+  err = 0;
+ }
+ mutex_unlock(&inode->i_mutex);
+ return err;
+}
+
diff --git a/fs/sysfs/file.c b/fs/sysfs/file.c
index fc46333..31b5429 100644
--- a/fs/sysfs/file.c
+++ b/fs/sysfs/file.c
@@ -606,7 +606,7 @@ EXPORT_SYMBOL_GPL(sysfs_chmod_file);

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


@@ -623,7 +623,7 @@ void sysfs_remove_file_from_group(struct kobject *kobj,
```

```
  dir = lookup_one_len(group, kobj->dentry, strlen(group));
  if (!IS_ERR(dir)) {
-  sysfs_hash_and_remove(dir, attr->name);
+  sysfs_hash_and_remove(kobj, dir, attr->name);
   dput(dir);
  }
 }
diff --git a/fs/sysfs/group.c b/fs/sysfs/group.c
index 46a277b..61115b9 100644
--- a/fs/sysfs/group.c
+++ b/fs/sysfs/group.c
@@ -17,16 +17,16 @@
 #include "sysfs.h"


-static void remove_files(struct dentry * dir,
+static void remove_files(struct kobject *kobj, struct dentry * dir,
     const struct attribute_group * grp)
 {
  struct attribute *const* attr;

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

-static int create_files(struct dentry * dir,
+static int create_files(struct kobject *kobj, struct dentry * dir,
    const struct attribute_group * grp)
 {
  struct attribute *const* attr;
@@ -36,7 +36,7 @@ static int create_files(struct dentry * dir,
   error = sysfs_add_file(dir, *attr, SYSFS_KOBJ_ATTR);
  }
  if (error)
-  remove_files(dir,grp);
+  remove_files(kobj, dir, grp);
  return error;
 }

@@ -56,7 +56,7 @@ int sysfs_create_group(struct kobject * kobj,
  } else
   dir = kobj->dentry;
  dir = dget(dir);
- if ((error = create_files(dir,grp))) {
+ if ((error = create_files(kobj, dir, grp))) {
   if (grp->name)
    sysfs_remove_subdir(dir);
```

```
 }
@@ -75,7 +75,7 @@ void sysfs_remove_group(struct kobject * kobj,
  else
   dir = dget(kobj->dentry);

- remove_files(dir,grp);
+ remove_files(kobj, dir, grp);
  if (grp->name)
   sysfs_remove_subdir(dir);
  /* release the ref. taken in this routine */
diff --git a/fs/sysfs/inode.c b/fs/sysfs/inode.c
index 69e6e9b..6b0e536 100644
--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -191,6 +191,7 @@ const unsigned char * sysfs_get_name(struct sysfs_dirent *sd)
  BUG_ON(!sd || !sd->s_element);

  switch (sd->s_type) {
+  case SYSFS_SHADOW:
   case SYSFS_DIR:
    /* Always have a dentry so use that */
    return sd->s_dentry->d_name.name;
@@ -259,8 +260,9 @@ void sysfs_drop_dentry(struct sysfs_dirent * sd, struct dentry * parent)
  }
 }

-int sysfs_hash_and_remove(struct dentry * dir, const char * name)
+int sysfs_hash_and_remove(struct kobject *kobj, struct dentry * dir, const char * name)
 {
+ struct inode * inode;
  struct sysfs_dirent * sd;
  struct sysfs_dirent * parent_sd;
  int found = 0;
@@ -272,8 +274,13 @@ int sysfs_hash_and_remove(struct dentry * dir, const char * name)
   /* no inode means this hasn't been made visible yet */
   return -ENOENT;

+ inode = dir->d_inode;
+ mutex_lock_nested(&inode->i_mutex, I_MUTEX_PARENT);
+ dir = sysfs_resolve_for_remove(kobj, dir);
+ if (IS_ERR(dir))
+  goto out_unlock;
+ dget(dir);
  parent_sd = dir->d_fsdata;
- mutex_lock_nested(&dir->d_inode->i_mutex, I_MUTEX_PARENT);
  list_for_each_entry(sd, &parent_sd->s_children, s_sibling) {
   if (!sd->s_element)
    continue;
```

```
@@ -285,7 +292,12 @@ int sysfs_hash_and_remove(struct dentry * dir, const char * name)
    break;
   }
  }
- mutex_unlock(&dir->d_inode->i_mutex);
+out_unlock:
+ mutex_unlock(&inode->i_mutex);
+ if (!IS_ERR(dir) && dir) {
+  sysfs_prune_shadow(dir);
+  dput(dir);
+ }

  return found ? 0 : -ENOENT;
 }
diff --git a/fs/sysfs/symlink.c b/fs/sysfs/symlink.c
index 7b9c5bf..2885ebb 100644
--- a/fs/sysfs/symlink.c
+++ b/fs/sysfs/symlink.c
@@ -84,7 +84,7 @@ exit1:
  */
 int sysfs_create_link(struct kobject * kobj, struct kobject * target, const char * name)
 {
- struct dentry *dentry = NULL;
+ struct dentry *dir, *dentry = NULL;
  int error = -EEXIST;

  BUG_ON(!name);
@@ -99,8 +99,11 @@ int sysfs_create_link(struct kobject * kobj, struct kobject * target, const
char
   return -EFAULT;

  mutex_lock(&dentry->d_inode->i_mutex);
- if (!sysfs_dirent_exist(dentry->d_fsdata, name))
-  error = sysfs_add_link(dentry, name, target);
+ dir = sysfs_resolve_for_create(target, dentry);
+ if (IS_ERR(dir))
+  error = PTR_ERR(dir);
+ else if (!sysfs_dirent_exist(dir->d_fsdata, name))
+  error = sysfs_add_link(dir, name, target);
  mutex_unlock(&dentry->d_inode->i_mutex);
  return error;
 }
@@ -114,7 +117,7 @@ int sysfs_create_link(struct kobject * kobj, struct kobject * target, const
char

 void sysfs_remove_link(struct kobject * kobj, const char * name)
 {
- sysfs_hash_and_remove(kobj->dentry,name);
```

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

 static int sysfs_get_target_path(struct kobject * kobj, struct kobject * target,
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index ce6b3ec..d761b28 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -13,6 +13,10 @@ struct sysfs_dirent {
 extern struct vfsmount * sysfs_mount;
 extern struct kmem_cache *sysfs_dir_cachep;

+extern void sysfs_prune_shadow(struct dentry *shadow);
+struct dentry *sysfs_resolve_for_create(struct kobject *, struct dentry *);
+struct dentry *sysfs_resolve_for_remove(struct kobject *, struct dentry *);
+
 extern struct inode * sysfs_new_inode(mode_t mode, struct sysfs_dirent *);
 extern int sysfs_create(struct dentry *, int mode, int (*init)(struct inode *));

@@ -21,7 +25,7 @@ extern int sysfs_make_dirent(struct sysfs_dirent *, struct dentry *, void *,
     umode_t, int);

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

 extern int sysfs_create_subdir(struct kobject *, const char *, struct dentry **);
@@ -44,6 +48,11 @@ struct sysfs_symlink {
  struct kobject * target_kobj;
 };

+struct sysfs_shadow_dir {
+ void *tag;
+ struct kobject *kobj;
+};
+
 struct sysfs_buffer {
  struct list_head  associates;
  size_t    count;
@@ -88,6 +97,9 @@ static inline struct kobject *sysfs_get_kobject(struct dentry *dentry)
   if (sd->s_type & SYSFS_KOBJ_LINK) {
    struct sysfs_symlink * sl = sd->s_element;
    kobj = kobject_get(sl->target_kobj);
+  } else if (sd->s_type & SYSFS_SHADOW) {
+   struct sysfs_shadow_dir * sdd = sd->s_element;
+   kobj = kobject_get(sdd->kobj);
   } else
```

```
    kobj = kobject_get(sd->s_element);
  }
@@ -104,6 +116,10 @@ static inline void release_sysfs_dirent(struct sysfs_dirent * sd)
   kobject_put(sl->target_kobj);
   kfree(sl);
  }
+ else if (sd->s_type & SYSFS_SHADOW) {
+  struct sysfs_shadow_dir * sdd = sd->s_element;
+  kfree(sdd);
+ }
  kfree(sd->s_iattr);
  kmem_cache_free(sysfs_dir_cachep, sd);
 }
diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
index 1f1bb45..cda7b81 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -68,11 +68,17 @@ struct sysfs_ops {
  ssize_t (*store)(struct kobject *,struct attribute *,const char *, size_t);
 };

+struct shadow_dir_operations {
+ void *(*current_tag)(void);
+ void *(*kobject_tag)(struct kobject *kobj);
+};
+
 #define SYSFS_ROOT  0x0001
 #define SYSFS_DIR  0x0002
 #define SYSFS_KOBJ_ATTR  0x0004
 #define SYSFS_KOBJ_BIN_ATTR 0x0008
 #define SYSFS_KOBJ_LINK  0x0020
+#define SYSFS_SHADOW  0x0040
 #define SYSFS_NOT_PINNED (SYSFS_KOBJ_ATTR | SYSFS_KOBJ_BIN_ATTR |
SYSFS_KOBJ_LINK)

 #ifdef CONFIG_SYSFS
@@ -124,6 +130,8 @@ void sysfs_remove_file_from_group(struct kobject *kobj,

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

+int sysfs_enable_shadowing(struct kobject *, const struct shadow_dir_operations *);
+
 extern int __must_check sysfs_init(void);

 #else /* CONFIG_SYSFS */
@@ -219,6 +227,12 @@ static inline void sysfs_notify(struct kobject * k, char *dir, char *attr)
 {
 }
```

```
+static inline int sysfs_enable_shadowing(struct kobject *kobj,
+    const struct shadow_dir_operations *shadow_ops)
+{
+ return 0;
+}
+
 static inline int __must_check sysfs_init(void)
 {
  return 0;
--
1.5.0.g53756
```

_____

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