Subject: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.
Posted by ebiederm on Thu, 19 Jul 2007 04:45:13 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
and the sysfs 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 | 404 ++--------
 fs/sysfs/file.c | 4 +-
 fs/sysfs/group.c | 12 +-
 fs/sysfs/inode.c | 11 +-
 fs/sysfs/symlink.c | 30 +++-
 fs/sysfs/sysfs.h | 9 +-
 include/linux/sysfs.h | 15 ++
 8 files changed, 396 insertions(+), 91 deletions(-)

diff --git a/fs/sysfs/bin.c b/fs/sysfs/bin.c
index 135353f..1ef0a07 100644
--- a/fs/sysfs/bin.c
+++ b/fs/sysfs/bin.c
@@ -248,7 +248,7 @@ 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->sd, attr->attr.name) < 0) {
+ if (sysfs_hash_and_remove(kobj, kobj->sd, 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 f88130c..e6d367e 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -14,12 +14,33 @@

```
 #include <asm/semaphore.h>
 #include "sysfs.h"

+static void sysfs_prune_shadow_sd(struct sysfs_dirent *sd);
+
 DEFINE_MUTEX(sysfs_mutex);
 spinlock_t sysfs_assoc_lock = SPIN_LOCK_UNLOCKED;

 static spinlock_t sysfs_ino_lock = SPIN_LOCK_UNLOCKED;
 static DEFINE_IDA(sysfs_ino_ida);

+static struct sysfs_dirent *find_shadow_sd(struct sysfs_dirent *parent_sd, const void *target)
+{
+ /* Find the shadow directory for the specified tag */
+ struct sysfs_dirent *sd;
+
+ for (sd = parent_sd->s_children; sd; sd = sd->s_sibling) {
+  if (sd->s_name != target)
+   continue;
+  break;
+ }
+ return sd;
+}
+
+static const void *find_shadow_tag(struct kobject *kobj)
+{
+ /* Find the tag the current kobj is cached with */
+ return kobj->sd->s_parent->s_name;
+}
+
 /**
  * sysfs_link_sibling - link sysfs_dirent into sibling list
  * @sd: sysfs_dirent of interest
@@ -323,7 +344,8 @@ void release_sysfs_dirent(struct sysfs_dirent * sd)
  if (sysfs_type(sd) & SYSFS_COPY_NAME)
   kfree(sd->s_name);
  kfree(sd->s_iattr);
- sysfs_free_ino(sd->s_ino);
+ if (sysfs_type(sd) != SYSFS_SHADOW_DIR)
+  sysfs_free_ino(sd->s_ino);
  kmem_cache_free(sysfs_dir_cachep, sd);

  sd = parent_sd;
@@ -414,7 +436,8 @@ static void sysfs_attach_dentry(struct sysfs_dirent *sd, struct dentry
*dentry)
 {
  sd->s_dentry = dentry;
  spin_unlock(&sysfs_assoc_lock);
```

```
- d_rehash(dentry);
+ if (dentry->d_flags & DCACHE_UNHASHED)
+  d_rehash(dentry);
 }

 static int sysfs_ilookup_test(struct inode *inode, void *arg)
@@ -569,6 +592,10 @@ static void sysfs_drop_dentry(struct sysfs_dirent *sd)
  spin_unlock(&dcache_lock);
  spin_unlock(&sysfs_assoc_lock);

+ /* dentries for shadowed directories are pinned, unpin */
+ if ((sysfs_type(sd) == SYSFS_SHADOW_DIR) ||
+     (sd->s_flags & SYSFS_FLAG_SHADOWED))
+  dput(dentry);
  dput(dentry);

  /* adjust nlink and update timestamp */
@@ -622,6 +649,7 @@ int sysfs_addrm_finish(struct sysfs_addrm_cxt *acxt)
   acxt->removed = sd->s_sibling;
   sd->s_sibling = NULL;

+  sysfs_prune_shadow_sd(sd->s_parent);
   sysfs_drop_dentry(sd);
   sysfs_deactivate(sd);
   sysfs_put(sd);
@@ -687,6 +715,7 @@ static int create_dir(struct kobject *kobj, struct sysfs_dirent *parent_sd,
  umode_t mode = S_IFDIR| S_IRWXU | S_IRUGO | S_IXUGO;
  struct sysfs_addrm_cxt acxt;
  struct sysfs_dirent *sd;
+ int err;

  /* allocate */
  sd = sysfs_new_dirent(name, mode, SYSFS_DIR);
@@ -696,15 +725,21 @@ static int create_dir(struct kobject *kobj, struct sysfs_dirent *parent_sd,

  /* link in */
  sysfs_addrm_start(&acxt, parent_sd);
+ err = -ENOENT;
+ if (!sysfs_resolve_for_create(kobj, &acxt.parent_sd))
+  goto addrm_finish;

- if (!sysfs_find_dirent(parent_sd, name)) {
+ err = -EEXIST;
+ if (!sysfs_find_dirent(acxt.parent_sd, name)) {
   sysfs_add_one(&acxt, sd);
   sysfs_link_sibling(sd);
+  err = 0;
  }
```

```
+addrm_finish:
  if (!sysfs_addrm_finish(&acxt)) {
   sysfs_put(sd);
-  return -EEXIST;
+  return err;
  }

  *p_sd = sd;
@@ -813,18 +848,56 @@ static struct dentry * sysfs_lookup(struct inode *dir, struct dentry
*dentry,
  return NULL;
 }

+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_flags & SYSFS_FLAG_SHADOWED) {
+  const struct shadow_dir_operations *shadow_ops;
+  const void *tag;
+
+  mutex_lock(&sysfs_mutex);
+
+  shadow_ops = dentry->d_inode->i_private;
+  tag = shadow_ops->current_tag();
+
+  sd = find_shadow_sd(sd, tag);
+  if (sd)
+   dest = sd->s_dentry;
+  dget(dest);
+
+  mutex_unlock(&sysfs_mutex);
+ }
+ if (!dest)
+  dest = dget(dentry);
+ dput(nd->dentry);
+ nd->dentry = dest;
+
+ return NULL;
+}
+
+
 const struct inode_operations sysfs_dir_inode_operations = {
  .lookup  = sysfs_lookup,
```

```
  .setattr = sysfs_setattr,
+ .follow_link = sysfs_shadow_follow_link,
 };

+static void __remove_dir(struct sysfs_addrm_cxt *acxt, struct sysfs_dirent *sd)
+{
+ sysfs_unlink_sibling(sd);
+ sysfs_remove_one(acxt, sd);
+}
+
 static void remove_dir(struct sysfs_dirent *sd)
 {
  struct sysfs_addrm_cxt acxt;

  sysfs_addrm_start(&acxt, sd->s_parent);
- sysfs_unlink_sibling(sd);
- sysfs_remove_one(&acxt, sd);
+ __remove_dir(&acxt, sd);
  sysfs_addrm_finish(&acxt);
 }

@@ -833,17 +906,11 @@ void sysfs_remove_subdir(struct sysfs_dirent *sd)
  remove_dir(sd);
 }

-
-static void __sysfs_remove_dir(struct sysfs_dirent *dir_sd)
+static void sysfs_empty_dir(struct sysfs_addrm_cxt *acxt,
+       struct sysfs_dirent *dir_sd)
 {
- struct sysfs_addrm_cxt acxt;
  struct sysfs_dirent **pos;

- if (!dir_sd)
-  return;
-
- pr_debug("sysfs %s: removing dir\n", dir_sd->s_name);
- sysfs_addrm_start(&acxt, dir_sd);
  pos = &dir_sd->s_children;
  while (*pos) {
   struct sysfs_dirent *sd = *pos;
@@ -851,10 +918,39 @@ static void __sysfs_remove_dir(struct sysfs_dirent *dir_sd)
   if (sysfs_type(sd) && sysfs_type(sd) != SYSFS_DIR) {
    *pos = sd->s_sibling;
    sd->s_sibling = NULL;
-   sysfs_remove_one(&acxt, sd);
+   sysfs_remove_one(acxt, sd);
   } else
```

```
    pos = &(*pos)->s_sibling;
  }
+}
+
+static void sysfs_remove_shadows(struct sysfs_addrm_cxt * acxt,
+      struct sysfs_dirent *dir_sd)
+{
+ struct sysfs_dirent **pos;
+
+ pos = &dir_sd->s_children;
+ while (*pos) {
+  struct sysfs_dirent *sd = *pos;
+
+  sysfs_empty_dir(acxt, sd);
+  __remove_dir(acxt, sd);
+ }
+}
+
+static void __sysfs_remove_dir(struct sysfs_dirent *dir_sd)
+{
+ struct sysfs_addrm_cxt acxt;
+
+ if (!dir_sd)
+  return;
+
+ pr_debug("sysfs %s: removing dir\n", dir_sd->s_name);
+ sysfs_addrm_start(&acxt, dir_sd);
+ if (sysfs_type(dir_sd) == SYSFS_DIR)
+  sysfs_empty_dir(&acxt, dir_sd);
+ else
+  sysfs_remove_shadows(&acxt, dir_sd);
  sysfs_addrm_finish(&acxt);

  remove_dir(dir_sd);
@@ -882,86 +978,75 @@ void sysfs_remove_dir(struct kobject * kobj)

 int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
 {
+ struct dentry *old_dentry, *new_dentry, *parent;
+ struct sysfs_addrm_cxt acxt;
  struct sysfs_dirent *sd;
- struct dentry *parent = NULL;
- struct dentry *old_dentry = NULL, *new_dentry = NULL;
- struct sysfs_dirent *parent_sd;
- const char *dup_name = NULL;
+ const char *dup_name;
  int error;
```

```
- if (!kobj->parent)
-  return -EINVAL;
+ dup_name = NULL;
+ new_dentry = NULL;

- /* get dentries */
  sd = kobj->sd;
- old_dentry = sysfs_get_dentry(sd);
- if (IS_ERR(old_dentry)) {
-  error = PTR_ERR(old_dentry);
-  goto out_dput;
- }
-
- parent_sd = kobj->parent->sd;
- parent = sysfs_get_dentry(parent_sd);
- if (IS_ERR(parent)) {
-  error = PTR_ERR(parent);
-  goto out_dput;
- }
-
- /* lock parent and get dentry for new name */
- mutex_lock(&parent->d_inode->i_mutex);
+ sysfs_addrm_start(&acxt, sd->s_parent);
+ error = -ENOENT;
+ if (!sysfs_resolve_for_create(kobj, &acxt.parent_sd))
+  goto addrm_finish;

- new_dentry = lookup_one_len(new_name, parent, strlen(new_name));
- if (IS_ERR(new_dentry)) {
-  error = PTR_ERR(new_dentry);
-  goto out_unlock;
- }
+ error = -EEXIST;
+ if (sysfs_find_dirent(acxt.parent_sd, new_name))
+  goto addrm_finish;

  error = -EINVAL;
- if (old_dentry == new_dentry)
-  goto out_unlock;
+ if ((sd->s_parent == acxt.parent_sd) &&
+     (strcmp(new_name, sd->s_name) == 0))
+  goto addrm_finish;
+
+ old_dentry = sd->s_dentry;
+ parent = acxt.parent_sd->s_dentry;
+ if (old_dentry) {
+  old_dentry = sd->s_dentry;
+  parent = acxt.parent_sd->s_dentry;
```

```
+  new_dentry = lookup_one_len(new_name, parent, strlen(new_name));
+  if (IS_ERR(new_dentry)) {
+   error = PTR_ERR(new_dentry);
+   goto addrm_finish;
+  }

- error = -EEXIST;
- if (new_dentry->d_inode)
-  goto out_unlock;
+  error = -EINVAL;
+  if (old_dentry == 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;

  dup_name = sd->s_name;
  sd->s_name = new_name;

  /* 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(parent_sd);
+ sysfs_get(acxt.parent_sd);
  sysfs_put(sd->s_parent);
- sd->s_parent = parent_sd;
+ sd->s_parent = acxt.parent_sd;
  sysfs_link_sibling(sd);

- mutex_unlock(&sysfs_mutex);
-
+ if (new_dentry) {
+  d_add(new_dentry, NULL);
+  d_move(old_dentry, new_dentry);
```

```
+ }
  error = 0;
- goto out_unlock;
+addrm_finish:
+ sysfs_addrm_finish(&acxt);

- out_drop:
- d_drop(new_dentry);
- out_unlock:
- mutex_unlock(&parent->d_inode->i_mutex);
- out_dput:
  kfree(dup_name);
- dput(parent);
- dput(old_dentry);
  dput(new_dentry);
  return error;
 }
@@ -1098,8 +1183,11 @@ static int sysfs_readdir(struct file * filp, void * dirent, filldir_t filldir)
    i++;
    /* fallthrough */
   default:
-   mutex_lock(&sysfs_mutex);
+   /* If I am the shadow master return nothing. */
+   if (parent_sd->s_flags & SYSFS_FLAG_SHADOWED)
+    return 0;

+   mutex_lock(&sysfs_mutex);
    pos = &parent_sd->s_children;
    while (*pos != cursor)
     pos = &(*pos)->s_sibling;
@@ -1188,3 +1276,185 @@ const struct file_operations sysfs_dir_operations = {
  .read  = generic_read_dir,
  .readdir = sysfs_readdir,
 };
+
+
+static void sysfs_prune_shadow_sd(struct sysfs_dirent *sd)
+{
+ struct sysfs_addrm_cxt acxt;
+
+ /* If a shadow directory goes empty remove it. */
+ if (sysfs_type(sd) != SYSFS_SHADOW_DIR)
+  return;
+
+ if (sd->s_children)
+  return;
+
+ sysfs_addrm_start(&acxt, sd->s_parent);
```

```
+
+ if (sd->s_flags & SYSFS_FLAG_REMOVED)
+  goto addrm_finish;
+
+ if (sd->s_children)
+  goto addrm_finish;
+
+ __remove_dir(&acxt, sd);
+addrm_finish:
+ sysfs_addrm_finish(&acxt);
+}
+
+static struct sysfs_dirent *add_shadow_sd(struct sysfs_dirent *parent_sd, const void *tag)
+{
+ struct sysfs_dirent *sd = NULL;
+ struct dentry *dir, *shadow;
+ struct inode *inode;
+
+ dir = parent_sd->s_dentry;
+ inode = dir->d_inode;
+
+ shadow = d_alloc(dir->d_parent, &dir->d_name);
+ if (!shadow)
+  goto out;
+
+ /* 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;
+
+ sd = sysfs_new_dirent(tag, parent_sd->s_mode, SYSFS_SHADOW_DIR);
+ if (!sd)
+  goto error;
+
+ sd->s_elem.dir.kobj = parent_sd->s_elem.dir.kobj;
+ sd->s_parent = sysfs_get(parent_sd);
+
+ /* Use the inode number of the parent we are shadowing */
+ sysfs_free_ino(sd->s_ino);
+ sd->s_ino = parent_sd->s_ino;
+
+ inc_nlink(inode);
+ inc_nlink(dir->d_parent->d_inode);
+
+ sysfs_link_sibling(sd);
+ __iget(inode);
```

```
+ sysfs_instantiate(shadow, inode);
+ sysfs_attach_dentry(sd, shadow);
+out:
+ return sd;
+error:
+ dput(shadow);
+ goto out;
+}
+
+int sysfs_resolve_for_create(struct kobject *kobj,
+    struct sysfs_dirent **parent_sd)
+{
+ const struct shadow_dir_operations *shadow_ops;
+ struct sysfs_dirent *sd, *shadow_sd;
+
+ sd = *parent_sd;
+ if (sysfs_type(sd) == SYSFS_SHADOW_DIR)
+  sd = sd->s_parent;
+
+ if (sd->s_flags & SYSFS_FLAG_SHADOWED) {
+  const void *tag;
+
+  shadow_ops = sd->s_dentry->d_inode->i_private;
+  tag = shadow_ops->kobject_tag(kobj);
+
+  shadow_sd = find_shadow_sd(sd, tag);
+  if (!shadow_sd)
+   shadow_sd = add_shadow_sd(sd, tag);
+  sd = shadow_sd;
+ }
+ if (sd) {
+  *parent_sd = sd;
+  return 1;
+ }
+ return 0;
+}
+
+int sysfs_resolve_for_remove(struct kobject *kobj,
+    struct sysfs_dirent **parent_sd)
+{
+ struct sysfs_dirent *sd;
+ /* If dentry is a shadow directory find the shadow that is
+  * stored under the same tag as kobj.  This allows removal
+  * of dirents to function properly even if the value of
+  * kobject_tag() has changed since we initially created
+  * the dirents assoctated with kobj.
+  */
+
```

```
+ sd = *parent_sd;
+ if (sysfs_type(sd) == SYSFS_SHADOW_DIR)
+  sd = sd->s_parent;
+ if (sd->s_flags & SYSFS_FLAG_SHADOWED) {
+  const void *tag;
+
+  tag = find_shadow_tag(kobj);
+  sd = find_shadow_sd(sd, tag);
+ }
+ if (sd) {
+  *parent_sd = sd;
+  return 1;
+ }
+ return 0;
+}
+
+/**
+ * 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;
+ int err;
+
+ /* Find the dentry for the shadowed directory and
+  * increase it's count.
+  */
+ err = -ENOENT;
```

```
+ sd = kobj->sd;
+ dentry = sysfs_get_dentry(sd);
+ if (!dentry)
+  goto out;
+
+ mutex_lock(&sysfs_mutex);
+ err = -EINVAL;
+ /* We can only enable shadowing on empty directories
+  * where shadowing is not already enabled.
+  */
+ if (!sd->s_children && (sysfs_type(sd) == SYSFS_DIR) &&
+     !(sd->s_flags & SYSFS_FLAG_REMOVED) &&
+     !(sd->s_flags & SYSFS_FLAG_SHADOWED)) {
+  sd->s_flags |= SYSFS_FLAG_SHADOWED;
+  dentry->d_inode->i_private = (void *)shadow_ops;
+  err = 0;
+ }
+ mutex_unlock(&sysfs_mutex);
+out:
+ if (err)
+  dput(dentry);
+ return err;
+}
+
diff --git a/fs/sysfs/file.c b/fs/sysfs/file.c
index 3e1cc06..530a830 100644
--- a/fs/sysfs/file.c
+++ b/fs/sysfs/file.c
@@ -544,7 +544,7 @@ 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);
 }


@@ -561,7 +561,7 @@ void sysfs_remove_file_from_group(struct kobject *kobj,

  dir_sd = sysfs_get_dirent(kobj->sd, group);
  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 4606f7c..9e928fd 100644
```

```
--- a/fs/sysfs/group.c
+++ b/fs/sysfs/group.c
@@ -17,16 +17,16 @@
 #include "sysfs.h"


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

  for (attr = grp->attrs; *attr; 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,
+static int create_files(struct kobject *kobj, struct sysfs_dirent *dir_sd,
    const struct attribute_group *grp)
 {
  struct attribute *const* attr;
@@ -35,7 +35,7 @@ static int create_files(struct sysfs_dirent *dir_sd,
   error = sysfs_add_file(dir_sd, *attr, SYSFS_KOBJ_ATTR);
  if (error)
-  remove_files(dir_sd, grp);
+  remove_files(kobj, dir_sd, grp);
  return error;
 }

@@ -55,7 +55,7 @@ int sysfs_create_group(struct kobject * kobj,
  } else
   sd = kobj->sd;
  sysfs_get(sd);
- error = create_files(sd, grp);
+ error = create_files(kobj, sd, grp);
  if (error) {
   if (grp->name)
    sysfs_remove_subdir(sd);
@@ -76,7 +76,7 @@ void sysfs_remove_group(struct kobject * kobj,
  } else
   sd = sysfs_get(dir_sd);

- remove_files(sd, grp);
+ remove_files(kobj, sd, grp);
  if (grp->name)
   sysfs_remove_subdir(sd);
```

```
diff --git a/fs/sysfs/inode.c b/fs/sysfs/inode.c
index 9671164..e31896e 100644
--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -187,17 +187,16 @@ void sysfs_instantiate(struct dentry *dentry, struct inode *inode)
  d_instantiate(dentry, 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 **pos, *sd;

- if (!dir_sd)
-  return -ENOENT;
-
  sysfs_addrm_start(&acxt, dir_sd);
+ if (!sysfs_resolve_for_remove(kobj, &acxt.parent_sd))
+  goto addrm_finish;

- for (pos = &dir_sd->s_children; *pos; pos = &(*pos)->s_sibling) {
+ for (pos = &acxt.parent_sd->s_children; *pos; pos = &(*pos)->s_sibling) {
   sd = *pos;

   if (!sysfs_type(sd))
@@ -209,7 +208,7 @@ int sysfs_hash_and_remove(struct sysfs_dirent *dir_sd, const char
*name)
    break;
   }
  }
-
+addrm_finish:
  if (sysfs_addrm_finish(&acxt))
   return 0;
  return -ENOENT;
diff --git a/fs/sysfs/symlink.c b/fs/sysfs/symlink.c
index 4ce687f..e52d3ef 100644
--- a/fs/sysfs/symlink.c
+++ b/fs/sysfs/symlink.c
@@ -15,8 +15,11 @@ static int object_depth(struct sysfs_dirent *sd)
 {
  int depth = 0;

- for (; sd->s_parent; sd = sd->s_parent)
+ for (; sd->s_parent; sd = sd->s_parent) {
+  if (sysfs_type(sd) == SYSFS_SHADOW_DIR)
```

```
+   continue;
   depth++;
+ }

  return depth;
 }
@@ -25,17 +28,24 @@ static int object_path_length(struct sysfs_dirent * sd)
 {
  int length = 1;

- for (; sd->s_parent; sd = sd->s_parent)
+ for (; sd->s_parent; sd = sd->s_parent) {
+  if (sysfs_type(sd) == SYSFS_SHADOW_DIR)
+   continue;
   length += strlen(sd->s_name) + 1;
+ }

  return length;
 }

 static void fill_object_path(struct sysfs_dirent *sd, char *buffer, int length)
 {
+ int cur;
  --length;
  for (; sd->s_parent; sd = sd->s_parent) {
-  int cur = strlen(sd->s_name);
+  if (sysfs_type(sd) == SYSFS_SHADOW_DIR)
+   continue;
+
+  cur = strlen(sd->s_name);

   /* back up enough to print this bus id with '/' */
   length -= cur;
@@ -91,16 +101,20 @@ int sysfs_create_link(struct kobject * kobj, struct kobject * target, const
char
  target_sd = NULL; /* reference is now owned by the symlink */

  sysfs_addrm_start(&acxt, parent_sd);
+ error = -ENOENT;
+ if (!sysfs_resolve_for_create(target, &acxt.parent_sd))
+  goto addrm_finish;

- if (!sysfs_find_dirent(parent_sd, name)) {
+ error = -EEXIST;
+ if (!sysfs_find_dirent(acxt.parent_sd, name)) {
+  error = 0;
   sysfs_add_one(&acxt, sd);
   sysfs_link_sibling(sd);
```

```
  }

- if (!sysfs_addrm_finish(&acxt)) {
-  error = -EEXIST;
+addrm_finish:
+ if (!sysfs_addrm_finish(&acxt))
   goto out_put;
- }

  return 0;

@@ -119,7 +133,7 @@ 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->sd, name);
+ sysfs_hash_and_remove(kobj, kobj->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 b55e510..ddfcef2 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -58,6 +58,12 @@ extern struct kmem_cache *sysfs_dir_cachep;
 extern struct dentry *sysfs_get_dentry(struct sysfs_dirent *sd);
 extern void sysfs_link_sibling(struct sysfs_dirent *sd);
 extern void sysfs_unlink_sibling(struct sysfs_dirent *sd);
+
+extern int sysfs_resolve_for_create(struct kobject *kobj,
+        struct sysfs_dirent **parent_sd);
+extern int sysfs_resolve_for_remove(struct kobject *kobj,
+        struct sysfs_dirent **parent_sd);
+
 extern struct sysfs_dirent *sysfs_get_active(struct sysfs_dirent *sd);
 extern void sysfs_put_active(struct sysfs_dirent *sd);
 extern struct sysfs_dirent *sysfs_get_active_two(struct sysfs_dirent *sd);
@@ -83,7 +89,8 @@ extern struct sysfs_dirent *sysfs_new_dirent(const char *name, umode_t
mode,

 extern int sysfs_add_file(struct sysfs_dirent *dir_sd,
      const struct attribute *attr, int type);
-extern int sysfs_hash_and_remove(struct sysfs_dirent *dir_sd, const char *name);
+extern int sysfs_hash_and_remove(struct kobject *kobj,
+     struct sysfs_dirent *dir_sd, const char *name);
 extern struct sysfs_dirent *sysfs_find(struct sysfs_dirent *dir, const char * name);
```

```
 extern int sysfs_create_subdir(struct kobject *kobj, const char *name,
diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
index c16e4c5..7cda047 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -74,16 +74,23 @@ struct sysfs_ops {
  ssize_t (*store)(struct kobject *,struct attribute *,const char *, size_t);
 };

+struct shadow_dir_operations {
+ const void *(*current_tag)(void);
+ const void *(*kobject_tag)(struct kobject *kobj);
+};
+
 #define SYSFS_TYPE_MASK  0x00ff
 #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_DIR 0x0040
 #define SYSFS_COPY_NAME  (SYSFS_DIR | SYSFS_KOBJ_LINK)

 #define SYSFS_FLAG_MASK  ~SYSFS_TYPE_MASK
 #define SYSFS_FLAG_REMOVED 0x0100
+#define SYSFS_FLAG_SHADOWED 0x0200

 #ifdef CONFIG_SYSFS

@@ -134,6 +141,8 @@ 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 */
@@ -229,6 +238,12 @@ 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;
--
1.5.1.1.181.g2de0
```

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