Subject: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by Benjamin Thery on Fri, 06 Jun 2008 15:47:54 GMT
View Forum Message <> Reply to Message

sysfs: Implement sysfs tagged directory support.

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 race free design all tagged directories are created
and managed by sysfs itself. The upper level code that knows what
tagged directories we need provides just two methods that enable
this:
sb_tag() - that returns a "void *" tag that identifies the context of
the process that mounted sysfs.
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 sb_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 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        |  185 ++++++++++++++++++++++++++++++++++++++++++++++----
 fs/sysfs/file.c       |    8 +-
 fs/sysfs/group.c      |    4 -
 fs/sysfs/inode.c      |    7 +
 fs/sysfs/mount.c      |   44 ++++++++++-
 fs/sysfs/symlink.c    |    2
 fs/sysfs/sysfs.h      |   17 ++++
 include/linux/sysfs.h |   17 ++++
 9 files changed, 257 insertions(+), 29 deletions(-)

Index: linux-mm/fs/sysfs/bin.c
===================================================================
--- linux-mm.orig/fs/sysfs/bin.c
+++ linux-mm/fs/sysfs/bin.c
@@ -252,7 +252,7 @@ int sysfs_create_bin_file(struct kobject

 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);
Index: linux-mm/fs/sysfs/dir.c
===================================================================
--- linux-mm.orig/fs/sysfs/dir.c
+++ linux-mm/fs/sysfs/dir.c
@@ -103,8 +103,17 @@ static void sysfs_unlink_sibling(struct
 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 && sd->s_parent->s_flags & SYSFS_FLAG_TAGGED) {
+  const void *tag;

```
+  tag = sysfs_lookup_tag(sd->s_parent, sb);
+  if (sd->s_tag.tag != tag)
+   return ERR_PTR(-EXDEV);
+ }

+ dentry = dget(sb->s_root);
  while (dentry->d_fsdata != sd) {
   struct sysfs_dirent *cur;
   struct dentry *parent;
@@ -423,7 +432,11 @@ void sysfs_addrm_start(struct sysfs_addr
  */
 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)) {
   printk(KERN_WARNING "sysfs: duplicate filename '%s' "
         "can not be created\n", sd->s_name);
   WARN_ON(1);
@@ -439,6 +452,9 @@ int sysfs_add_one(struct sysfs_addrm_cxt

  sd->s_parent = sysfs_get(acxt->parent_sd);

+ if (sd->s_parent->s_flags & SYSFS_FLAG_TAGGED)
+  sd->s_tag.tag = tag;
+
  if (sysfs_type(sd) == SYSFS_DIR && acxt->parent_inode)
   inc_nlink(acxt->parent_inode);

@@ -585,13 +601,18 @@ void sysfs_addrm_finish(struct sysfs_add
  * 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 ((parent_sd->s_flags & SYSFS_FLAG_TAGGED) &&
+      (sd->s_tag.tag != tag))
+   continue;
   if (!strcmp(sd->s_name, name))
    return sd;
```

```
+ }
  return NULL;
 }

@@ -615,7 +636,7 @@ struct sysfs_dirent *sysfs_get_dirent(st
  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);

@@ -681,13 +702,16 @@ static struct dentry * sysfs_lookup(stru
     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;
+ const void *tag;

  mutex_lock(&sysfs_mutex);

- sd = sysfs_find_dirent(parent_sd, dentry->d_name.name);
+ tag = sysfs_lookup_tag(parent_sd, parent->d_sb);
+ sd = sysfs_find_dirent(parent_sd, tag, dentry->d_name.name);

  /* no such entry */
  if (!sd) {
@@ -895,19 +919,24 @@ int sysfs_rename_dir(struct kobject * ko
  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 = sysfs_dirent_tag(sd);
+ 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);
@@ -920,7 +949,7 @@ int sysfs_rename_dir(struct kobject * ko
  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 kobject and sysfs_dirent */
@@ -935,6 +964,8 @@ int sysfs_rename_dir(struct kobject * ko

  dup_name = sd->s_name;
  sd->s_name = new_name;
+ if (sd->s_parent->s_flags & SYSFS_FLAG_TAGGED)
+  sd->s_tag.tag = tag;

  /* rename */
  list_for_each_entry(srs, &todo, list) {
@@ -942,6 +973,20 @@ int sysfs_rename_dir(struct kobject * ko
   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);
@@ -964,11 +1009,13 @@ int sysfs_move_dir(struct kobject *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 = sysfs_dirent_tag(sd);

  error = 0;
  if (sd->s_parent == new_parent_sd)
@@ -1002,7 +1049,7 @@ 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;
@@ -1041,10 +1088,11 @@ static inline unsigned char dt_type(stru

 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;
+ const void *tag;

  if (filp->f_pos == 0) {
   ino = parent_sd->s_ino;
@@ -1062,6 +1110,8 @@ static int sysfs_readdir(struct file * f
  if ((filp->f_pos > 1) && (filp->f_pos < INT_MAX)) {
   mutex_lock(&sysfs_mutex);

+  tag = sysfs_lookup_tag(parent_sd, parent->d_sb);
+
   /* Skip the dentries we have already reported */
```

```
   pos = parent_sd->s_dir.children;
   while (pos && (filp->f_pos > pos->s_ino))
@@ -1071,6 +1121,10 @@ static int sysfs_readdir(struct file * f
    const char * name;
    int len;

+   if ((parent_sd->s_flags & SYSFS_FLAG_TAGGED) &&
+       (pos->s_tag.tag != tag))
+    continue;
+
    name = pos->s_name;
    len = strlen(name);
    filp->f_pos = ino = pos->s_ino;
@@ -1091,3 +1145,106 @@ const struct file_operations sysfs_dir_o
  .read  = generic_read_dir,
  .readdir = sysfs_readdir,
 };
+
+const void *sysfs_creation_tag(struct sysfs_dirent *parent_sd,
+          struct sysfs_dirent *sd)
+{
+ const void *tag = NULL;
+
+ if (parent_sd->s_flags & SYSFS_FLAG_TAGGED) {
+  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 = parent_sd->s_tag.ops->kobject_tag(kobj);
+ }
+ return tag;
+}
+
+const void *sysfs_removal_tag(struct kobject *kobj, struct sysfs_dirent *dir_sd)
+{
+ const void *tag = NULL;
+
+ if (dir_sd->s_flags & SYSFS_FLAG_TAGGED)
+  tag = kobj->sd->s_tag.tag;
+
+ return tag;
```

```
+}
+
+const void *sysfs_lookup_tag(struct sysfs_dirent *dir_sd,
+        struct super_block *sb)
+{
+ const void *tag = NULL;
+
+ if (dir_sd->s_flags & SYSFS_FLAG_TAGGED)
+  tag = dir_sd->s_tag.ops->sb_tag(&sysfs_info(sb)->tag);
+
+ return tag;
+}
+
+const void *sysfs_dirent_tag(struct sysfs_dirent *sd)
+{
+ const void *tag = NULL;
+
+ if (sd->s_parent && (sd->s_parent->s_flags & SYSFS_FLAG_TAGGED))
+  tag = sd->s_tag.tag;
+
+ return tag;
+}
+
+/**
+ * sysfs_enable_tagging - Automatically tag all of the children 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.
+ *
+ * tag_ops->sb_tag() returns the context for a given superblock.
+ *
+ * tag_ops->kobject_tag() returns the context that a given kobj
+ * resides in.
+ *
+ * Using those methods the sysfs code on tagged 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_tagging(struct kobject *kobj,
+ const struct sysfs_tagged_dir_operations *tag_ops)
+{
```

```
+ struct sysfs_dirent *sd;
+ int err;
+
+ err = -ENOENT;
+ sd = kobj->sd;
+
+ mutex_lock(&sysfs_mutex);
+ err = -EINVAL;
+ /* We can only enable tagging on empty directories
+  * where tagging is not already enabled, and
+  * who are not subdirectories of directories where tagging is
+  * enabled.
+  */
+ if (!sd->s_dir.children && (sysfs_type(sd) == SYSFS_DIR) &&
+     !(sd->s_flags & SYSFS_FLAG_TAGGED) &&
+     sd->s_parent &&
+     !(sd->s_parent->s_flags & SYSFS_FLAG_TAGGED)) {
+  err = 0;
+  sd->s_flags |= SYSFS_FLAG_TAGGED;
+  sd->s_tag.ops = tag_ops;
+ }
+ mutex_unlock(&sysfs_mutex);
+ return err;
+}
Index: linux-mm/fs/sysfs/file.c
===================================================================
--- linux-mm.orig/fs/sysfs/file.c
+++ linux-mm/fs/sysfs/file.c
@@ -460,9 +460,9 @@ void sysfs_notify(struct kobject *k, cha
  mutex_lock(&sysfs_mutex);

  if (sd && dir)
-  sd = sysfs_find_dirent(sd, dir);
+  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;

@@ -631,7 +631,7 @@ 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);
 }
```

```
@@ -651,7 +651,7 @@ void sysfs_remove_file_from_group(struct
  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);
  }
 }
Index: linux-mm/fs/sysfs/group.c
===================================================================
--- linux-mm.orig/fs/sysfs/group.c
+++ linux-mm/fs/sysfs/group.c
@@ -23,7 +23,7 @@ static void remove_files(struct sysfs_di
  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_dir
    * 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)
Index: linux-mm/fs/sysfs/inode.c
===================================================================
--- linux-mm.orig/fs/sysfs/inode.c
+++ linux-mm/fs/sysfs/inode.c
@@ -217,17 +217,20 @@ struct inode * sysfs_get_inode(struct sy
  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 = sysfs_removal_tag(kobj, dir_sd);

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

Index: linux-mm/fs/sysfs/mount.c
===================================================================
--- linux-mm.orig/fs/sysfs/mount.c
+++ linux-mm/fs/sysfs/mount.c
@@ -75,6 +75,7 @@ static int sysfs_fill_super(struct super
   goto out_err;
  }
  root->d_fsdata = &sysfs_root;
+ root->d_sb = sb;
  sb->s_root = root;
  sb->s_fs_info = info;
  return 0;
@@ -88,20 +89,55 @@ out_err:
  return error;
 }

+static int sysfs_test_super(struct super_block *sb, void *ptr)
+{
+ struct task_struct *task = ptr;
+ struct sysfs_super_info *info = sysfs_info(sb);
+ int found = 1;
+
+ 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;
+ struct super_block *sb;
+ int error;
  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, current);
+ 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);
+  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)
Index: linux-mm/fs/sysfs/symlink.c
===================================================================
--- linux-mm.orig/fs/sysfs/symlink.c
+++ linux-mm/fs/sysfs/symlink.c
@@ -94,7 +94,7 @@ void sysfs_remove_link(struct kobject *
  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,
```

```
Index: linux-mm/fs/sysfs/sysfs.h
===================================================================
--- linux-mm.orig/fs/sysfs/sysfs.h
+++ linux-mm/fs/sysfs/sysfs.h
@@ -46,6 +46,10 @@ struct sysfs_dirent {
  const char  *s_name;

  union {
+  const struct sysfs_tagged_dir_operations *ops;
+  const void      *tag;
+ } s_tag;
+ union {
   struct sysfs_elem_dir  s_dir;
   struct sysfs_elem_symlink s_symlink;
   struct sysfs_elem_attr  s_attr;
@@ -69,6 +73,7 @@ struct sysfs_dirent {

 #define SYSFS_FLAG_MASK   ~SYSFS_TYPE_MASK
 #define SYSFS_FLAG_REMOVED  0x0200
+#define SYSFS_FLAG_TAGGED  0x0400

 static inline unsigned int sysfs_type(struct sysfs_dirent *sd)
 {
@@ -87,6 +92,7 @@ struct sysfs_addrm_cxt {

 struct sysfs_super_info {
  int grabbed;
+ struct sysfs_tag_info tag;
 };

 #define sysfs_info(SB) ((struct sysfs_super_info *)(SB)->s_fs_info)
@@ -113,6 +119,13 @@ extern spinlock_t sysfs_assoc_lock;
 extern const struct file_operations sysfs_dir_operations;
 extern const struct inode_operations sysfs_dir_inode_operations;

+extern const void *sysfs_creation_tag(struct sysfs_dirent *parent_sd,
+          struct sysfs_dirent *sd);
+extern const void *sysfs_removal_tag(struct kobject *kobj,
+         struct sysfs_dirent *dir_sd);
+extern const void *sysfs_lookup_tag(struct sysfs_dirent *dir_sd,
+        struct super_block *sb);
+extern const void *sysfs_dirent_tag(struct sysfs_dirent *sd);
 struct dentry *sysfs_get_dentry(struct super_block *sb,
     struct sysfs_dirent *sd);
 struct sysfs_dirent *sysfs_get_active_two(struct sysfs_dirent *sd);
@@ -124,6 +137,7 @@ void sysfs_remove_one(struct sysfs_addrm
 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 +169,8 @@ static inline void sysfs_put(struct sysf
  */
 struct inode *sysfs_get_inode(struct sysfs_dirent *sd);
 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);

 /*
Index: linux-mm/include/linux/sysfs.h
===================================================================
--- linux-mm.orig/include/linux/sysfs.h
+++ linux-mm/include/linux/sysfs.h
@@ -78,6 +78,14 @@ struct sysfs_ops {
  ssize_t (*store)(struct kobject *,struct attribute *,const char *, size_t);
 };

+struct sysfs_tag_info {
+};
+
+struct sysfs_tagged_dir_operations {
+ const void *(*sb_tag)(struct sysfs_tag_info *info);
+ const void *(*kobject_tag)(struct kobject *kobj);
+};
+
 #ifdef CONFIG_SYSFS

 int sysfs_schedule_callback(struct kobject *kobj, void (*func)(void *),
@@ -117,6 +125,9 @@ void sysfs_remove_file_from_group(struct
 void sysfs_notify(struct kobject *kobj, char *dir, char *attr);
 void sysfs_printk_last_file(void);

+int sysfs_enable_tagging(struct kobject *kobj,
+    const struct sysfs_tagged_dir_operations *tag_ops);
+
 extern int __must_check sysfs_init(void);

 #else /* CONFIG_SYSFS */
@@ -213,6 +224,12 @@ static inline void sysfs_notify(struct k
 {
 }
```

```
+static inline int sysfs_enable_tagging(struct kobject *kobj,
+         const struct sysfs_tagged_dir_operations *tag_ops)
+{
+ return 0;
+}
+
 static inline int __must_check sysfs_init(void)
 {
  return 0;

--
```

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