Subject: [netns] sysfs: issues porting shadow directories on top of 2.6.21-mm2
Posted by Benjamin Thery on Mon, 18 Jun 2007 17:57:44 GMT
View Forum Message <> Reply to Message

Hi Eric,

For the past few weeks, I've been trying to port your netns patchset
on top of 2.6.21-mm2. It took me a lot more time than I first expected
to have something working.

I started the port based on your latest public git repository tag:
"netns/v2.6.21-rc6-netns17".

I met a few difficulties during the port the worst being porting the
shadow directories patches on top of Greg's sysfs patches.

Greg modified a lot of things in sysfs and I had to "rewrite"/adapt
most of your "sysfs: Implement sysfs manged shadow directory support"
patch. My knowledge of sysfs approaching zero, the result isn't that
great.

Any chance you've updated the patchset for a recent version of the -mm
kernel?

Here are some issues I have with the sysfs part of the netns patchset:

* The first thing I'm not sure to understand in your patch is how
shadow dirs and there "real" counterpart are supposed to be linked
(via dentry and via sysfs_dirent).

Is it something like:

/sys/class/net/ ("real" net class)
/sys/class/net-shadow1/
/sys/class/net-shadow2/

or:

/sys/class/net/
/sys/class/net/net-shadow1/
/sys/class/net/net-shadow2/

In add_shadow_dir(), it seems the shadow dentry parent is "class" :
  shadow = d_alloc(dir->d_parent, &dir->d_name);
and the shadow sysfs_dirent parent is the real "net":
  sysfs_make_dirent(dir->d_fsdata, ...);

On 2.6.21-mm2, if I attach the dentry to "class" (dir->d_parent) as

you did initially, then the shadow directory lookup "fails": it
always returns the same shadow dir, whatever network namespace is
current. Indeed, sysfs_shadow_follow_link() is never called with a
SYSFS_DIR dentry, but always directly with a SYSFS_SHADOW one, and the
tag comparison is never done.

In add_shadow_dir(), I changed the d_alloc() call and passed dir
instead of dir->d_parent, and it "solved" the issue:
sysfs_shadow_follow_link() is called with the SYSFS_DIR dentry, and
the shadow dir lookup is done.

* I also have some issues with symlinks.

Indeed, the way symlinks are "resolved" have changed. Symlinks paths
aren't resolved anymore using kobject linking but uses sysfs_dirent
instead. So I had to use a dirty hack to skip shadow directories in
fs/sysfs/symlink.c: fill_object_path()/object_path_length().

Regards.
Benjamin

--
B e n j a m i n T h e r y - BULL/DT/Open Software R&D

http://www.bull.com

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

---

Subject: Re: [netns] [PATCH 1/4] sysfs: porting shadow directories on top of 2.6.21-mm2
Posted by Benjamin Thery on Mon, 18 Jun 2007 18:06:19 GMT
View Forum Message <> Reply to Message

Here are the patches for the sysfs shadow directories ported to
2.6.21-mm2 in case you want to look at them to spot the issues.
They are extracted from the full netns patchset.

Benjamin

--
B e n j a m i n T h e r y - BULL/DT/Open Software R&D

http://www.bull.com

_____

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

---

Subject: Re: [netns] [PATCH 2/4] sysfs: issues porting shadow directories on top of 2.6.21-mm2
Posted by Benjamin Thery on Mon, 18 Jun 2007 18:08:16 GMT
View Forum Message <> Reply to Message

This is the big piece.

Benjamin

_____

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

---

Subject: Re: [netns] [PATCH 3/4] sysfs: issues porting shadow directories on top of 2.6.21-mm2
Posted by Benjamin Thery on Mon, 18 Jun 2007 18:09:16 GMT
View Forum Message <> Reply to Message

Third one.

Benjamin

_____

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

---

Subject: Re: [netns] sysfs: issues porting shadow directories on top of 2.6.21-mm2
Posted by Benjamin Thery on Mon, 18 Jun 2007 18:14:04 GMT
View Forum Message <> Reply to Message

Last one, that contains the dirty hack that "fixes" symlinks through
shadow directories in my pacthset.

If you have time to help me port/fix these patches that would be great.
If you already have an updated version pf the patchset that would be

even better.

Thanks for your help.
Benjamin

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

---

Subject: Re: [netns] sysfs: issues porting shadow directories on top of 2.6.21-mm2
Posted by ebiederm on Tue, 19 Jun 2007 17:32:05 GMT
View Forum Message <> Reply to Message

Benjamin Thery <benjamin.thery@bull.net> writes:

> Hi Eric,
>
> For the past few weeks, I've been trying to port your netns patchset on top of
> 2.6.21-mm2. It took me a lot more time than I first expected to have something
> working.
>
> I started the port based on your latest public git repository tag:
> "netns/v2.6.21-rc6-netns17".
>
> I met a few difficulties during the port the worst being porting the shadow
> directories patches on top of Greg's sysfs patches.
>
> Greg modified a lot of things in sysfs and I had to "rewrite"/adapt most of your
> "sysfs: Implement sysfs manged shadow directory support" patch. My knowledge of
> sysfs approaching zero, the result isn't that great.

Grrr. I will try and take a look shortly.

> Any chance you've updated the patchset for a recent version of the -mm kernel?

I haven't I will have to take a look. It sounds like more has changed then
I anticipated.

> Here are some issues I have with the sysfs part of the netns patchset:
>
> * The first thing I'm not sure to understand in your patch is how shadow dirs
> and there "real" counterpart are supposed to be linked (via dentry and via
> sysfs_dirent).
>
> Is it something like:
>

> /sys/class/net/ ("real" net class)
> /sys/class/net-shadow1/
> /sys/class/net-shadow2/

Yes. At least as far as dentry are concerned.

> or:
>
> /sys/class/net/
> /sys/class/net/net-shadow1/
> /sys/class/net/net-shadow2/

Not as far as dentries are concerned.

The basic concept is that you get a different result depending who you are.

> In add_shadow_dir(), it seems the shadow dentry parent is "class" :
> shadow = d_alloc(dir->d_parent, &dir->d_name);
> and the shadow sysfs_dirent parent is the real "net":
> sysfs_make_dirent(dir->d_fsdata, ...);
>
> On 2.6.21-mm2, if I attach the dentry to "class" (dir->d_parent) as you did
> initially, then the shadow directory lookup "fails": it always returns the same
> shadow dir, whatever network namespace is current. Indeed,
> sysfs_shadow_follow_link() is never called with a SYSFS_DIR dentry, but always
> directly with a SYSFS_SHADOW one, and the tag comparison is never done.
>
> In add_shadow_dir(), I changed the d_alloc() call and passed dir instead of
> dir->d_parent, and it "solved" the issue: sysfs_shadow_follow_link() is called
> with the SYSFS_DIR dentry, and the shadow dir lookup is done.
>
>
> * I also have some issues with symlinks.
>
> Indeed, the way symlinks are "resolved" have changed. Symlinks paths aren't
> resolved anymore using kobject linking but uses sysfs_dirent instead. So I had
> to use a dirty hack to skip shadow directories in fs/sysfs/symlink.c:
> fill_object_path()/object_path_length().

Thanks for the heads up. I will take a look

Eric

Subject: Re: [netns] sysfs: issues porting shadow directories on top of 2.6.21-mm2
Posted by ebiederm on Wed, 20 Jun 2007 00:19:06 GMT
View Forum Message <> Reply to Message

Benjamin Thery <benjamin.thery@bull.net> writes:

> Hi Eric,
>
> For the past few weeks, I've been trying to port your netns patchset on top of
> 2.6.21-mm2. It took me a lot more time than I first expected to have something
> working.

Ok. Interesting. There are a few pieces missing to make it work on the latest -mm.
The removal of the doubly linked lists..

> I started the port based on your latest public git repository tag:
> "netns/v2.6.21-rc6-netns17".
>
> I met a few difficulties during the port the worst being porting the shadow
> directories patches on top of Greg's sysfs patches.
>
> Greg modified a lot of things in sysfs and I had to "rewrite"/adapt most of your
> "sysfs: Implement sysfs manged shadow directory support" patch. My knowledge of
> sysfs approaching zero, the result isn't that great.
>
> Any chance you've updated the patchset for a recent version of the -mm kernel?

I'm looking at it.

> Here are some issues I have with the sysfs part of the netns patchset:
>
> * The first thing I'm not sure to understand in your patch is how shadow dirs
> and there "real" counterpart are supposed to be linked (via dentry and via
> sysfs_dirent).
>
> Is it something like:
>
> /sys/class/net/ ("real" net class)
> /sys/class/net-shadow1/
> /sys/class/net-shadow2/
>
> or:
>
> /sys/class/net/
> /sys/class/net/net-shadow1/
> /sys/class/net/net-shadow2/

In the pure sysfs dirent data structures it does look like this
so I don't think your ``hack'' patch is a hack.

> In add_shadow_dir(), it seems the shadow dentry parent is "class" :
> shadow = d_alloc(dir->d_parent, &dir->d_name);
> and the shadow sysfs_dirent parent is the real "net":
> sysfs_make_dirent(dir->d_fsdata, ...);
>
> On 2.6.21-mm2, if I attach the dentry to "class" (dir->d_parent) as you did
> initially, then the shadow directory lookup "fails": it always returns the same
> shadow dir, whatever network namespace is current. Indeed,
> sysfs_shadow_follow_link() is never called with a SYSFS_DIR dentry, but always
> directly with a SYSFS_SHADOW one, and the tag comparison is never done.

> In add_shadow_dir(), I changed the d_alloc() call and passed dir instead of
> dir->d_parent, and it "solved" the issue: sysfs_shadow_follow_link() is called
> with the SYSFS_DIR dentry, and the shadow dir lookup is done.

Hmm.

> * I also have some issues with symlinks.
>
> Indeed, the way symlinks are "resolved" have changed.

Yes. The they resolve to sysfs_dirents instead of kobjects. From an implementation standpoint it should not make a big difference.

> Symlinks paths aren't
> resolved anymore using kobject linking but uses sysfs_dirent instead. So I had
> to use a dirty hack to skip shadow directories in fs/sysfs/symlink.c:
> fill_object_path()/object_path_length().

I'm not certain it is that dirty but yes. That change is needed.

Eric

Subject: Re: [netns] sysfs: issues porting shadow directories on top of 2.6.21-mm2
Posted by Dave Hansen on Thu, 21 Jun 2007 17:22:57 GMT
View Forum Message <> Reply to Message

On Mon, 2007-06-18 at 19:57 +0200, Benjamin Thery wrote:
> /sys/class/net/ ("real" net class)
> /sys/class/net-shadow1/
> /sys/class/net-shadow2/

This seems like a nice "quick fix", but do we really want to be hacking
sysfs around like this?

We have the backing sysfs_* entries that are separate from the vfs
entities already. Why can't we simply have different /sys vfsmounts
with different views of the backing sysfs_* entries?

-- Dave

Subject: Re: [netns] sysfs: issues porting shadow directories on top of 2.6.21-mm2
Posted by ebiederm on Thu, 21 Jun 2007 20:48:10 GMT
View Forum Message <> Reply to Message

Dave Hansen <hansendc@us.ibm.com> writes:

> On Mon, 2007-06-18 at 19:57 +0200, Benjamin Thery wrote:
>> /sys/class/net/   ("real" net class)
>> /sys/class/net-shadow1/
>> /sys/class/net-shadow2/
>
> This seems like a nice "quick fix", but do we really want to be hacking
> sysfs around like this?
>
> We have the backing sysfs_* entries that are separate from the vfs
> entities already. Why can't we simply have different /sys vfsmounts
> with different views of the backing sysfs_* entries?

So far this has been easier. sysfs is so tightly coupled to the kobject
tree decoupling them is seriously non-trivial.

When I can I do prefer separate mounts because that fixes a lot of cache
issues. For /proc we may be able to go as far as breaking up all of
it's pieces into separate filesystems, and do magic mounts and umounts
behind the users back. We can't even contemplate that with sysfs as
it has symlinks all over the place. A remount of sysfs that displays
just the information from the current processes namespaces would be
possible. If you can figure how to make those mods to sysfs I will
be happy to use it.

We actually must modify /proc to do multiple mounts because a follow
link method on the root directory entry is not honored. With a magic

follow link method we can use the dcache as intended except for the handful of magic directories.

So since I have working code today (baring one or two bugs I'm in the process of fixing) and that greg has given his basic nod of acceptance I figure I will pursue this route and we can update things later if we need to.

Eric

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

---

Subject: Re: [netns] sysfs: issues porting shadow directories on top of 2.6.21-mm2
Posted by ebiederm on Thu, 21 Jun 2007 21:02:38 GMT
View Forum Message <> Reply to Message

Dave Hansen <hansendc@us.ibm.com> writes:

> On Mon, 2007-06-18 at 19:57 +0200, Benjamin Thery wrote:
>> /sys/class/net/ ("real" net class)
>> /sys/class/net-shadow1/
>> /sys/class/net-shadow2/
>
> This seems like a nice "quick fix", but do we really want to be hacking
> sysfs around like this?
>
> We have the backing sysfs_* entries that are separate from the vfs
> entities already. Why can't we simply have different /sys vfsmounts
> with different views of the backing sysfs_* entries?

Dave. Another and probably better way of looking at this is we need essentially the same code in sysfs however we implement it. The only practical difference is do we get the namespaces from the mount of sysfs or from the current process doing the reading.

If you would like to take this in the direction of looking at the mount point for the information and seeing if there are optimizations we can apply with static information free.

Wherever we end up I think this code is a fairly decent starting place.

Eric

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [netns] sysfs: issues porting shadow directories on top of 2.6.21-mm2
Posted by gregkh on Thu, 21 Jun 2007 21:16:37 GMT
View Forum Message <> Reply to Message

On Thu, Jun 21, 2007 at 02:48:10PM -0600, Eric W. Biederman wrote:
> Dave Hansen <hansendc@us.ibm.com> writes:
>
> > On Mon, 2007-06-18 at 19:57 +0200, Benjamin Thery wrote:
> >> /sys/class/net/ ("real" net class)
> >> /sys/class/net-shadow1/
> >> /sys/class/net-shadow2/
> >
> > This seems like a nice "quick fix", but do we really want to be hacking
> > sysfs around like this?
> >
> > We have the backing sysfs_* entries that are separate from the vfs
> > entities already. Why can't we simply have different /sys vfsmounts
> > with different views of the backing sysfs_* entries?
>
> So far this has been easier. sysfs is so tightly coupled to the kobject
> tree decoupling them is seriously non-trivial.

Tejun just decoupled them, that's why this all changed in the -mm tree,
so it might be a whole lot easier to do now.

thanks,

greg k-h

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

Subject: Re: [netns] sysfs: issues porting shadow directories on top of 2.6.21-mm2
Posted by Dave Hansen on Thu, 21 Jun 2007 21:27:13 GMT
View Forum Message <> Reply to Message

On Thu, 2007-06-21 at 15:02 -0600, Eric W. Biederman wrote:
> Dave. Another and probably better way of looking at this is we need
> essentially the same code in sysfs however we implement it. The only
> practical difference is do we get the namespaces from the mount of
> sysfs or from the current process doing the reading.

Yeah, both approaches would definitely have the same result: changing
the view of /sys based on process context.

I'll look at it in a bit more detail and see if it is practical.

-- Dave

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

---

Subject: Re: [netns] sysfs: issues porting shadow directories on top of 2.6.21-mm2
Posted by ebiederm on Thu, 21 Jun 2007 21:54:13 GMT
View Forum Message <> Reply to Message

Greg KH <gregkh@suse.de> writes:

> On Thu, Jun 21, 2007 at 02:48:10PM -0600, Eric W. Biederman wrote:
>> Dave Hansen <hansendc@us.ibm.com> writes:
>>
>> > On Mon, 2007-06-18 at 19:57 +0200, Benjamin Thery wrote:
>> >> /sys/class/net/ ("real" net class)
>> >> /sys/class/net-shadow1/
>> >> /sys/class/net-shadow2/
>> >
>> > This seems like a nice "quick fix", but do we really want to be hacking
>> > sysfs around like this?
>> >
>> > We have the backing sysfs_* entries that are separate from the vfs
>> > entities already. Why can't we simply have different /sys vfsmounts
>> > with different views of the backing sysfs_* entries?
>>
>> So far this has been easier. sysfs is so tightly coupled to the kobject
>> tree decoupling them is seriously non-trivial.
>
> Tejun just decoupled them, that's why this all changed in the -mm tree,
> so it might be a whole lot easier to do now.

Right. And I was able to simply the code by using more extensively using
the improved sysfs_dirent. However we still have kobj->dentry. Which
makes a one to many situation tricky.

Eric

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [netns] sysfs: issues porting shadow directories on top of 2.6.21-mm2
Posted by gregkh on Fri, 22 Jun 2007 00:13:28 GMT
View Forum Message <> Reply to Message

On Thu, Jun 21, 2007 at 03:54:13PM -0600, Eric W. Biederman wrote:
> Greg KH <gregkh@suse.de> writes:
>
> > On Thu, Jun 21, 2007 at 02:48:10PM -0600, Eric W. Biederman wrote:
> >> Dave Hansen <hansendc@us.ibm.com> writes:
> >>
> >> > On Mon, 2007-06-18 at 19:57 +0200, Benjamin Thery wrote:
> >> >> /sys/class/net/ ("real" net class)
> >> >> /sys/class/net-shadow1/
> >> >> /sys/class/net-shadow2/
> >> >
> >> > This seems like a nice "quick fix", but do we really want to be hacking
> >> > sysfs around like this?
> >> >
> >> > We have the backing sysfs_* entries that are separate from the vfs
> >> > entities already. Why can't we simply have different /sys vfsmounts
> >> > with different views of the backing sysfs_* entries?
> >>
> >> So far this has been easier. sysfs is so tightly coupled to the kobject
> >> tree decoupling them is seriously non-trivial.
> >
> > Tejun just decoupled them, that's why this all changed in the -mm tree,
> > so it might be a whole lot easier to do now.
>
> Right. And I was able to simply the code by using more extensively using
> the improved sysfs_dirent. However we still have kobj->dentry. Which
> makes a one to many situation tricky.

Ah, yeah, that is still there. Well, any suggestions you might have for
removing that limitation (if it is one), is appreciated.

thanks,

greg k-h

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

Subject: Re: [netns] sysfs: issues porting shadow directories on top of 2.6.21-mm2
Posted by ebiederm on Fri, 22 Jun 2007 07:00:00 GMT
View Forum Message <> Reply to Message

Greg KH <gregkh@suse.de> writes:

> Ah, yeah, that is still there. Well, any suggestions you might have for
> removing that limitation (if it is one), is appreciated.

There can be some benefit in mounting filesystems so you can see what
another user sees. So I think it is worth looking at.

However all of the user that care are at best CONFIG_EXPERIMENTAL
for the next while.

Therefore as long as we have something that is close we can fix
the last couple of details after we have the bulk of the code
changes made.

Eric

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

Subject: [PATCH 1/4] sysfs: Remove first pass at shadow directory support
Posted by ebiederm on Fri, 22 Jun 2007 07:33:42 GMT
View Forum Message <> Reply to Message

While shadow directories appear to be a good idea, the current scheme
of controlling their creation and destruction outside of sysfs appears
to be a locking and maintenance nightmare in the face of sysfs
directories dynamically coming and going. Which can now occur for
directories containing network devices when CONFIG_SYSFS_DEPRECATED is
not set.

This patch removes everything from the initial shadow directory
support that allowed the shadow directory creation to be controlled at
a higher level. So except for a few bits of sysfs_rename_dir
everything from commit b592fcfe7f06c15ec11774b5be7ce0de3aa86e73 is now
gone.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

These patches are against 2.6.22-rc4-mm2 Hopefully that is new enough
to catch all of the in flight sysfs patches.

```
 fs/sysfs/dir.c          |  172 +++++++++++-----------------------------------
 fs/sysfs/group.c        |    1 -
 fs/sysfs/inode.c        |   10 ---
 fs/sysfs/mount.c        |    2 +-
 fs/sysfs/sysfs.h        |    6 --
 include/linux/kobject.h |    4 -
 include/linux/sysfs.h   |   24 +------
 lib/kobject.c           |   44 ++++----------
 8 files changed, 48 insertions(+), 215 deletions(-)

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index b4074ad..b1da4fc 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -415,10 +415,9 @@ int sysfs_create_subdir(struct kobject * k, const char * n, struct dentry **
d)
 /**
  * sysfs_create_dir - create a directory for an object.
  * @kobj:  object we're creating directory for.
- * @shadow_parent: parent parent object.
  */

-int sysfs_create_dir(struct kobject * kobj, struct dentry *shadow_parent)
+int sysfs_create_dir(struct kobject * kobj)
 {
  struct dentry * dentry = NULL;
  struct dentry * parent;
@@ -426,9 +425,7 @@ int sysfs_create_dir(struct kobject * kobj, struct dentry *shadow_parent)

  BUG_ON(!kobj);

- if (shadow_parent)
-  parent = shadow_parent;
- else if (kobj->parent)
+ if (kobj->parent)
   parent = kobj->parent->dentry;
  else if (sysfs_mount && sysfs_mount->mnt_sb)
   parent = sysfs_mount->mnt_sb->s_root;
@@ -516,12 +513,26 @@ void sysfs_remove_subdir(struct dentry * d)
 }


-static void __sysfs_remove_dir(struct dentry *dentry)
+/**
+ * sysfs_remove_dir - remove an object's directory.
+ * @kobj: object.
+ *
+ * The only thing special about this is that we remove any files in
```

```
+ * the directory before we remove the directory, and we've inlined
+ * what used to be sysfs_rmdir() below, instead of calling separately.
+ */
+
+void sysfs_remove_dir(struct kobject * kobj)
 {
+ struct dentry *dentry = kobj->dentry;
  struct sysfs_dirent *removed = NULL;
  struct sysfs_dirent *parent_sd;
  struct sysfs_dirent **pos;

+ spin_lock(&kobj_sysfs_assoc_lock);
+ kobj->dentry = NULL;
+ spin_unlock(&kobj_sysfs_assoc_lock);
+
  if (!dentry)
   return;

@@ -555,55 +566,35 @@ static void __sysfs_remove_dir(struct dentry *dentry)
  remove_dir(dentry);
 }

-/**
- * sysfs_remove_dir - remove an object's directory.
- * @kobj: object.
- *
- * The only thing special about this is that we remove any files in
- * the directory before we remove the directory, and we've inlined
- * what used to be sysfs_rmdir() below, instead of calling separately.
- */
-
-void sysfs_remove_dir(struct kobject * kobj)
-{
- struct dentry *d = kobj->dentry;
-
- spin_lock(&kobj_sysfs_assoc_lock);
- kobj->dentry = NULL;
- spin_unlock(&kobj_sysfs_assoc_lock);
-
- __sysfs_remove_dir(d);
-}
-
-int sysfs_rename_dir(struct kobject * kobj, struct dentry *new_parent,
-       const char *new_name)
+int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
 {
- struct sysfs_dirent *sd = kobj->dentry->d_fsdata;
- struct sysfs_dirent *parent_sd = new_parent->d_fsdata;
```

```
- struct dentry *new_dentry;
+ struct sysfs_dirent *sd;
+ struct sysfs_dirent *parent_sd;
+ struct inode *inode;
+ struct dentry *new_dentry, *parent;
  char *dup_name;
  int error;

- if (!new_parent)
-  return -EFAULT;
+ if (!kobj->parent)
+  return -EINVAL;
+
+ parent = dget(kobj->dentry->d_parent);
+ inode = parent->d_inode;
+
+ sd = kobj->dentry->d_fsdata;

  down_write(&sysfs_rename_sem);
- mutex_lock(&new_parent->d_inode->i_mutex);
+ mutex_lock(&inode->i_mutex);

- new_dentry = lookup_one_len(new_name, new_parent, strlen(new_name));
+ parent_sd = parent->d_fsdata;
+ new_dentry = lookup_one_len(new_name, parent_sd->s_dentry, strlen(new_name));
  if (IS_ERR(new_dentry)) {
   error = PTR_ERR(new_dentry);
   goto out_unlock;
  }

- /* By allowing two different directories with the same
-  * d_parent we allow this routine to move between different
-  * shadows of the same directory
-  */
  error = -EINVAL;
- if (kobj->dentry->d_parent->d_inode != new_parent->d_inode ||
-     new_dentry->d_parent->d_inode != new_parent->d_inode ||
-     new_dentry == kobj->dentry)
+ if (new_dentry == kobj->dentry)
   goto out_dput;

  error = -EEXIST;
@@ -643,8 +634,11 @@ int sysfs_rename_dir(struct kobject * kobj, struct dentry *new_parent,
  out_dput:
  dput(new_dentry);
  out_unlock:
- mutex_unlock(&new_parent->d_inode->i_mutex);
+ mutex_unlock(&inode->i_mutex);
```

```
  up_write(&sysfs_rename_sem);
+
+ dput(parent);
+
  return error;
 }

@@ -837,98 +831,6 @@ static loff_t sysfs_dir_lseek(struct file * file, loff_t offset, int origin)
  return offset;
 }

-
-/**
- * sysfs_make_shadowed_dir - Setup so a directory can be shadowed
- * @kobj: object we're creating shadow of.
- */
-
-int sysfs_make_shadowed_dir(struct kobject *kobj,
- void * (*follow_link)(struct dentry *, struct nameidata *))
-{
- struct inode *inode;
- struct inode_operations *i_op;
-
- inode = kobj->dentry->d_inode;
- if (inode->i_op != &sysfs_dir_inode_operations)
-  return -EINVAL;
-
- i_op = kmalloc(sizeof(*i_op), GFP_KERNEL);
- if (!i_op)
-  return -ENOMEM;
-
- memcpy(i_op, &sysfs_dir_inode_operations, sizeof(*i_op));
- i_op->follow_link = follow_link;
-
- /* Locking of inode->i_op?
-  * Since setting i_op is a single word write and they
-  * are atomic we should be ok here.
-  */
- inode->i_op = i_op;
- return 0;
-}
-
-/**
- * sysfs_create_shadow_dir - create a shadow directory for an object.
- * @kobj: object we're creating directory for.
- *
- * sysfs_make_shadowed_dir must already have been called on this
- * directory.
```

```
- */
-
-struct dentry *sysfs_create_shadow_dir(struct kobject *kobj)
-{
- struct dentry *dir = kobj->dentry;
- struct inode *inode = dir->d_inode;
- struct dentry *parent = dir->d_parent;
- struct sysfs_dirent *parent_sd = parent->d_fsdata;
- struct dentry *shadow;
- struct sysfs_dirent *sd;
-
- shadow = ERR_PTR(-EINVAL);
- if (!sysfs_is_shadowed_inode(inode))
-  goto out;
-
- shadow = d_alloc(parent, &dir->d_name);
- if (!shadow)
-  goto nomem;
-
- sd = sysfs_new_dirent("_SHADOW_", inode->i_mode, SYSFS_DIR);
- if (!sd)
-  goto nomem;
- sd->s_elem.dir.kobj = kobj;
- /* point to parent_sd but don't attach to it */
- sd->s_parent = sysfs_get(parent_sd);
- sysfs_attach_dirent(sd, NULL, shadow);
-
- d_instantiate(shadow, igrab(inode));
- inc_nlink(inode);
- inc_nlink(parent->d_inode);
-
- dget(shadow);  /* Extra count - pin the dentry in core */
-
-out:
- return shadow;
-nomem:
- dput(shadow);
- shadow = ERR_PTR(-ENOMEM);
- goto out;
-}
-
-/**
- * sysfs_remove_shadow_dir - remove an object's directory.
- * @shadow: dentry of shadow directory
- *
- * The only thing special about this is that we remove any files in
- * the directory before we remove the directory, and we've inlined
- * what used to be sysfs_rmdir() below, instead of calling separately.
```

```
- */
-
-void sysfs_remove_shadow_dir(struct dentry *shadow)
-{
- __sysfs_remove_dir(shadow);
-}
-
 const struct file_operations sysfs_dir_operations = {
  .open  = sysfs_dir_open,
  .release = sysfs_dir_close,
diff --git a/fs/sysfs/group.c b/fs/sysfs/group.c
index 52eed2a..82e0f55 100644
--- a/fs/sysfs/group.c
+++ b/fs/sysfs/group.c
@@ -13,7 +13,6 @@
 #include <linux/dcache.h>
 #include <linux/namei.h>
 #include <linux/err.h>
-#include <linux/fs.h>
 #include <asm/semaphore.h>
 #include "sysfs.h"

diff --git a/fs/sysfs/inode.c b/fs/sysfs/inode.c
index b1a4527..8181c49 100644
--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -34,16 +34,6 @@ static const struct inode_operations sysfs_inode_operations ={
  .setattr = sysfs_setattr,
 };

-void sysfs_delete_inode(struct inode *inode)
-{
- /* Free the shadowed directory inode operations */
- if (sysfs_is_shadowed_inode(inode)) {
-  kfree(inode->i_op);
-  inode->i_op = NULL;
- }
- return generic_delete_inode(inode);
-}
-
 int sysfs_setattr(struct dentry * dentry, struct iattr * iattr)
 {
  struct inode * inode = dentry->d_inode;
diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index 4be9593..c8126ab 100644
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -21,7 +21,7 @@ struct kmem_cache *sysfs_dir_cachep;
```

```
 static const struct super_operations sysfs_ops = {
  .statfs  = simple_statfs,
- .drop_inode = sysfs_delete_inode,
+ .drop_inode = generic_delete_inode,
 };

 static struct sysfs_dirent sysfs_root = {
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index 6f8aaf3..6258462 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -53,7 +53,6 @@ extern struct sysfs_dirent *sysfs_get_active_two(struct sysfs_dirent *sd);
 extern void sysfs_put_active_two(struct sysfs_dirent *sd);
 extern void sysfs_deactivate(struct sysfs_dirent *sd);

-extern void sysfs_delete_inode(struct inode *inode);
 extern void sysfs_init_inode(struct sysfs_dirent *sd, struct inode *inode);
 extern struct inode * sysfs_get_inode(struct sysfs_dirent *sd);
 extern void sysfs_instantiate(struct dentry *dentry, struct inode *inode);
@@ -100,8 +99,3 @@ static inline void sysfs_put(struct sysfs_dirent * sd)
  if (sd && atomic_dec_and_test(&sd->s_count))
   release_sysfs_dirent(sd);
 }
-
-static inline int sysfs_is_shadowed_inode(struct inode *inode)
-{
- return S_ISDIR(inode->i_mode) && inode->i_op->follow_link;
-}
diff --git a/include/linux/kobject.h b/include/linux/kobject.h
index c288e41..4fa5b88 100644
--- a/include/linux/kobject.h
+++ b/include/linux/kobject.h
@@ -71,13 +71,9 @@ extern void kobject_init(struct kobject *);
 extern void kobject_cleanup(struct kobject *);

 extern int __must_check kobject_add(struct kobject *);
-extern int __must_check kobject_shadow_add(struct kobject *, struct dentry *);
 extern void kobject_del(struct kobject *);

 extern int __must_check kobject_rename(struct kobject *, const char *new_name);
-extern int __must_check kobject_shadow_rename(struct kobject *kobj,
-      struct dentry *new_parent,
-      const char *new_name);
 extern int __must_check kobject_move(struct kobject *, struct kobject *);

 extern int __must_check kobject_register(struct kobject *);
diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
```

```
index 161e19a..a3fa5b9 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -17,8 +17,6 @@

 struct kobject;
 struct module;
-struct nameidata;
-struct dentry;

 /* FIXME
  * The *owner field is no longer used, but leave around
@@ -88,13 +86,13 @@ extern int sysfs_schedule_callback(struct kobject *kobj,
   void (*func)(void *), void *data, struct module *owner);

 extern int __must_check
-sysfs_create_dir(struct kobject *, struct dentry *);
+sysfs_create_dir(struct kobject *);

 extern void
 sysfs_remove_dir(struct kobject *);

 extern int __must_check
-sysfs_rename_dir(struct kobject *, struct dentry *, const char *new_name);
+sysfs_rename_dir(struct kobject *, const char *new_name);

 extern int __must_check
 sysfs_move_dir(struct kobject *, struct kobject *);
@@ -131,12 +129,6 @@ void sysfs_remove_file_from_group(struct kobject *kobj,

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

-
-extern int sysfs_make_shadowed_dir(struct kobject *kobj,
- void * (*follow_link)(struct dentry *, struct nameidata *));
-extern struct dentry *sysfs_create_shadow_dir(struct kobject *kobj);
-extern void sysfs_remove_shadow_dir(struct dentry *dir);
-
 extern int __must_check sysfs_init(void);

 #else /* CONFIG_SYSFS */
@@ -147,7 +139,7 @@ static inline int sysfs_schedule_callback(struct kobject *kobj,
  return -ENOSYS;
 }

-static inline int sysfs_create_dir(struct kobject * k, struct dentry *shadow)
+static inline int sysfs_create_dir(struct kobject * k)
 {
```

```
 return 0;
}
@@ -157,9 +149,7 @@ static inline void sysfs_remove_dir(struct kobject * k)
 ;
}

-static inline int sysfs_rename_dir(struct kobject * k,
-     struct dentry *new_parent,
-     const char *new_name)
+static inline int sysfs_rename_dir(struct kobject * k, const char *new_name)
{
 return 0;
}
@@ -234,12 +224,6 @@ static inline void sysfs_notify(struct kobject * k, char *dir, char *attr)
{
}

-static inline int sysfs_make_shadowed_dir(struct kobject *kobj,
- void * (*follow_link)(struct dentry *, struct nameidata *))
-{
- return 0;
-}
-
static inline int __must_check sysfs_init(void)
{
 return 0;
diff --git a/lib/kobject.c b/lib/kobject.c
index cb40a00..738d76f 100644
--- a/lib/kobject.c
+++ b/lib/kobject.c
@@ -46,11 +46,11 @@ static int populate_dir(struct kobject * kobj)
 return error;
}

-static int create_dir(struct kobject * kobj, struct dentry *shadow_parent)
+static int create_dir(struct kobject * kobj)
{
 int error = 0;
 if (kobject_name(kobj)) {
-  error = sysfs_create_dir(kobj, shadow_parent);
+  error = sysfs_create_dir(kobj);
  if (!error) {
   if ((error = populate_dir(kobj)))
    sysfs_remove_dir(kobj);
@@ -201,12 +201,11 @@ static void unlink(struct kobject * kobj)
}

/**
```

```
- * kobject_shadow_add - add an object to the hierarchy.
+ * kobject_add - add an object to the hierarchy.
  * @kobj: object.
- * @shadow_parent: sysfs directory to add to.
  */

-int kobject_shadow_add(struct kobject * kobj, struct dentry *shadow_parent)
+int kobject_add(struct kobject * kobj)
 {
  int error = 0;
  struct kobject * parent;
@@ -238,7 +237,7 @@ int kobject_shadow_add(struct kobject * kobj, struct dentry
*shadow_parent)
   kobj->parent = parent;
  }

- error = create_dir(kobj, shadow_parent);
+ error = create_dir(kobj);
  if (error) {
   /* unlink does the kobject_put() for us */
   unlink(kobj);
@@ -260,16 +259,6 @@ int kobject_shadow_add(struct kobject * kobj, struct dentry
*shadow_parent)
 }

 /**
- * kobject_add - add an object to the hierarchy.
- * @kobj: object.
- */
-int kobject_add(struct kobject * kobj)
-{
- return kobject_shadow_add(kobj, NULL);
-}
-
-
-/**
  * kobject_register - initialize and add an object.
  * @kobj: object in question.
  */
@@ -382,7 +371,7 @@ int kobject_rename(struct kobject * kobj, const char *new_name)
  /* Note : if we want to send the new name alone, not the full path,
   * we could probably use kobject_name(kobj); */

- error = sysfs_rename_dir(kobj, kobj->parent->dentry, new_name);
+ error = sysfs_rename_dir(kobj, new_name);

  /* This function is mostly/only used for network interface.
   * Some hotplug package track interfaces by their name and
```

```
@@ -399,27 +388,6 @@ out:
 }

 /**
- * kobject_rename - change the name of an object
- * @kobj: object in question.
- * @new_parent: object's new parent
- * @new_name: object's new name
- */
-
-int kobject_shadow_rename(struct kobject * kobj, struct dentry *new_parent,
-     const char *new_name)
-{
- int error = 0;
-
- kobj = kobject_get(kobj);
- if (!kobj)
-  return -EINVAL;
- error = sysfs_rename_dir(kobj, new_parent, new_name);
- kobject_put(kobj);
-
- return error;
-}
-
-/**
  * kobject_move - move object to another parent
  * @kobj: object in question.
  * @new_parent: object's new parent (can be NULL)
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.
Posted by ebiederm on Fri, 22 Jun 2007 07:35:51 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 | 347 ++-----
 fs/sysfs/file.c | 4 +-
 fs/sysfs/group.c | 12 +-
 fs/sysfs/inode.c | 15 ++-
 fs/sysfs/symlink.c | 26 +++-
 fs/sysfs/sysfs.h | 6 +-
 include/linux/sysfs.h | 14 ++
 8 files changed, 374 insertions(+), 52 deletions(-)

diff --git a/fs/sysfs/bin.c b/fs/sysfs/bin.c
index 3c5574a..b96a893 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->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 b1da4fc..043464e 100644
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -302,7 +302,8 @@ static void sysfs_attach_dentry(struct sysfs_dirent *sd, struct dentry
*dentry)
  sd->s_dentry = dentry;
  spin_unlock(&sysfs_lock);

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

 void sysfs_attach_dirent(struct sysfs_dirent *sd,
@@ -348,12 +349,17 @@ static int create_dir(struct kobject *kobj, struct dentry *parent,

```
  umode_t mode = S_IFDIR| S_IRWXU | S_IRUGO | S_IXUGO;
  struct dentry *dentry;
  struct inode *inode;
- struct sysfs_dirent *sd;
+ struct sysfs_dirent *sd, *parent_sd;

  mutex_lock(&parent->d_inode->i_mutex);

  /* allocate */
- dentry = lookup_one_len(name, parent, strlen(name));
+ parent_sd = sysfs_resolve_for_create(kobj, parent);
+ if (IS_ERR(parent_sd)) {
+  error = PTR_ERR(parent_sd);
+  goto out_unlock;
+ }
+ dentry = lookup_one_len(name, parent_sd->s_dentry, strlen(name));
  if (IS_ERR(dentry)) {
   error = PTR_ERR(dentry);
   goto out_unlock;
@@ -382,12 +388,12 @@ static int create_dir(struct kobject *kobj, struct dentry *parent,

  /* link in */
  error = -EEXIST;
- if (sysfs_dirent_exist(parent->d_fsdata, name))
+ if (sysfs_dirent_exist(parent_sd, name))
   goto out_iput;

  sysfs_instantiate(dentry, inode);
- inc_nlink(parent->d_inode);
- sysfs_attach_dirent(sd, parent->d_fsdata, dentry);
+ inc_nlink(parent_sd->s_dentry->d_inode);
+ sysfs_attach_dirent(sd, parent_sd, dentry);

  *p_dentry = dentry;
  error = 0;
@@ -502,9 +508,11 @@ static void remove_dir(struct dentry * d)

  mutex_unlock(&parent->d_inode->i_mutex);

+ sysfs_prune_shadow_sd(sd->s_parent);
  sysfs_drop_dentry(sd);
  sysfs_deactivate(sd);
  sysfs_put(sd);
+
 }

 void sysfs_remove_subdir(struct dentry * d)
@@ -512,6 +520,64 @@ void sysfs_remove_subdir(struct dentry * d)
```

```
  remove_dir(d);
 }

+static void sysfs_empty_dir(struct dentry *dentry,
+    struct sysfs_dirent **removed)
+{
+ struct sysfs_dirent *parent_sd;
+ struct sysfs_dirent **pos;
+
+ parent_sd = dentry->d_fsdata;
+ pos = &parent_sd->s_children;
+ while (*pos) {
+  struct sysfs_dirent *sd = *pos;
+
+  if (sd->s_type && (sd->s_type & SYSFS_NOT_PINNED)) {
+   *pos = sd->s_sibling;
+   sd->s_sibling = *removed;
+   *removed = sd;
+  } else
+   pos = &(*pos)->s_sibling;
+ }
+}
+
+static void __sysfs_remove_empty_shadow(struct dentry *shadow)
+{
+ struct sysfs_dirent *sd;
+
+ if (d_unhashed(shadow))
+  return;
+
+ sd = shadow->d_fsdata;
+ sysfs_unlink_sibling(sd);
+ simple_rmdir(shadow->d_inode, shadow);
+ d_delete(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 **removed)
+{
+ struct sysfs_dirent *parent_sd, **pos;
```

```
+
+ parent_sd = dentry->d_fsdata;
+ pos = &parent_sd->s_children;
+ while (*pos) {
+  struct sysfs_dirent *sd = *pos;
+  struct dentry *shadow;
+
+  shadow = dget(sd->s_dentry);
+  sysfs_empty_dir(shadow, removed);
+  __sysfs_remove_empty_shadow(shadow);
+  dput(shadow);
+ }
+}

 /**
  * sysfs_remove_dir - remove an object's directory.
@@ -524,10 +590,8 @@ void sysfs_remove_subdir(struct dentry * d)

 void sysfs_remove_dir(struct kobject * kobj)
 {
- struct dentry *dentry = kobj->dentry;
+ struct dentry *dentry = dget(kobj->dentry);
  struct sysfs_dirent *removed = NULL;
- struct sysfs_dirent *parent_sd;
- struct sysfs_dirent **pos;

  spin_lock(&kobj_sysfs_assoc_lock);
  kobj->dentry = NULL;
@@ -538,18 +602,10 @@ void sysfs_remove_dir(struct kobject * kobj)

  pr_debug("sysfs %s: removing dir\n",dentry->d_name.name);
  mutex_lock(&dentry->d_inode->i_mutex);
- parent_sd = dentry->d_fsdata;
- pos = &parent_sd->s_children;
- while (*pos) {
-  struct sysfs_dirent *sd = *pos;
-
-  if (sd->s_type && (sd->s_type & SYSFS_NOT_PINNED)) {
-   *pos = sd->s_sibling;
-   sd->s_sibling = removed;
-   removed = sd;
-  } else
-   pos = &(*pos)->s_sibling;
- }
+ if (dentry->d_inode->i_op == &sysfs_dir_inode_operations)
+  sysfs_empty_dir(dentry, &removed);
+ else
+  sysfs_remove_shadows(dentry, &removed);
```

```
  mutex_unlock(&dentry->d_inode->i_mutex);

  while (removed) {
@@ -586,7 +642,7 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  down_write(&sysfs_rename_sem);
  mutex_lock(&inode->i_mutex);

- parent_sd = parent->d_fsdata;
+ parent_sd = sysfs_resolve_for_create(kobj, parent);
  new_dentry = lookup_one_len(new_name, parent_sd->s_dentry, strlen(new_name));
  if (IS_ERR(new_dentry)) {
   error = PTR_ERR(new_dentry);
@@ -637,6 +693,7 @@ int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
  mutex_unlock(&inode->i_mutex);
  up_write(&sysfs_rename_sem);

+ sysfs_prune_shadow_sd(parent->d_fsdata);
  dput(parent);

  return error;
@@ -644,25 +701,36 @@ 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_sd = new_parent_dentry->d_fsdata;
+ new_parent_sd = sysfs_resolve_for_create(kobj, new_parent_dentry);
+ if (IS_ERR(new_parent_sd)) {
+  error = PTR_ERR(new_parent_sd);
+  goto out;
+ }
+
+ new_parent_dentry = new_parent_sd->s_dentry;
  sd = kobj->dentry->d_fsdata;

  new_dentry = lookup_one_len(kobj->name, new_parent_dentry,
@@ -684,8 +752,11 @@ again:
  sysfs_link_sibling(sd);

 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_sd(old_parent_dentry->d_fsdata);
+ dput(old_parent_dentry);

  return error;
 }
@@ -754,6 +825,11 @@ static int sysfs_readddir(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;
+
    pos = &parent_sd->s_children;
    while (*pos != cursor)
     pos = &(*pos)->s_sibling;
@@ -838,3 +914,212 @@ const struct file_operations sysfs_dir_operations = {
  .read  = generic_read_dir,
  .readdir = sysfs_readdir,
 };
+
+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 */
+ struct sysfs_dirent *sd;
+
+ sd = kobj->dentry->d_fsdata;
+ return sd->s_parent->s_name;
+}
+
+static struct sysfs_dirent *add_shadow_sd(struct sysfs_dirent *parent_sd, const void *tag)
+{
+ struct sysfs_dirent *sd;
+ struct dentry *dir, *shadow;
+ struct inode *inode;
+ int err;
+
+ dir = parent_sd->s_dentry;
+ inode = dir->d_inode;
+
+ err = -ENOMEM;
+ shadow = d_alloc(dir->d_parent, &dir->d_name);
+ if (!shadow)
+  goto error;
+
+ sd = sysfs_new_dirent(tag, inode->i_mode, SYSFS_SHADOW);
+ if (!sd)
+  goto error;
+
+ /* 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;
+
+ sysfs_instantiate(shadow, igrab(inode));
```

```
+ inc_nlink(inode);
+ inc_nlink(dir->d_parent->d_inode);
+ sysfs_attach_dirent(sd, parent_sd, shadow);
+out:
+ return sd;
+error:
+ dput(shadow);
+ sd = ERR_PTR(err);
+ goto out;
+}
+
+void sysfs_prune_shadow_sd(struct sysfs_dirent *sd)
+{
+ /* If a shadow directory goes empty remove it. */
+ struct dentry *shadow;
+
+ if (sd->s_type != SYSFS_SHADOW)
+  return;
+
+ shadow = sd->s_dentry;
+ if (!shadow)
+  return;
+
+ if (sd->s_children)
+  return;
+
+ sysfs_remove_empty_shadow(shadow);
+}
+
+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);
+  sd = find_shadow_sd(sd, shadow_ops->current_tag());
+  if (sd)
+   dest = sd->s_dentry;
+  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,
+};
+
+
+struct sysfs_dirent *sysfs_resolve_for_create(struct kobject *kobj,
+           struct dentry *dentry)
+{
+ const struct shadow_dir_operations *shadow_ops;
+ struct sysfs_dirent *sd, *shadow_sd;
+ struct inode *inode;
+
+ sd = dentry->d_fsdata;
+ inode = dentry->d_inode;
+ shadow_ops = inode->i_private;
+ if (inode->i_op == &sysfs_shadow_inode_operations) {
+  const void *tag;
+  if (sd->s_type == SYSFS_SHADOW)
+   sd = sd->s_parent;
+  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;
+ }
+ return sd;
+}
+
+struct sysfs_dirent *sysfs_resolve_for_remove(struct kobject *kobj, struct dentry *dentry)
+{
+ /* 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.
+  */
+ const struct shadow_dir_operations *shadow_ops;
+ struct sysfs_dirent *sd;
```

```
+ struct inode *inode;
+
+ sd = dentry->d_fsdata;
+ inode = dentry->d_inode;
+ shadow_ops = inode->i_private;
+ if (inode->i_op == &sysfs_shadow_inode_operations) {
+  if (sd->s_type == SYSFS_SHADOW)
+   sd = sd->s_parent;
+  sd = find_shadow_sd(sd, find_shadow_tag(kobj));
+  if (!sd)
+   sd = ERR_PTR(-ENOENT);
+ }
+ return sd;
+}
+
+/**
+ * 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 (!sd->s_children && (sd->s_type == SYSFS_DIR) &&
+     (inode->i_op == &sysfs_dir_inode_operations)) {
+  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 a84b734..8509ae1 100644
--- a/fs/sysfs/file.c
+++ b/fs/sysfs/file.c
@@ -560,7 +560,7 @@ 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);
 }


@@ -577,7 +577,7 @@ 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 82e0f55..0ae49ab 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);
  }
@@ -77,7 +77,7 @@ 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 8181c49..2ac07fd 100644
--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -272,10 +272,11 @@ void sysfs_drop_dentry(struct sysfs_dirent *sd)
   dput(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 **pos, *sd;
- struct sysfs_dirent *parent_sd = dir->d_fsdata;
+ struct sysfs_dirent *parent_sd;
  int found = 0;

  if (!dir)
@@ -285,7 +286,11 @@ int sysfs_hash_and_remove(struct dentry * dir, const char * name)
   /* no inode means this hasn't been made visible yet */
   return -ENOENT;

- mutex_lock_nested(&dir->d_inode->i_mutex, I_MUTEX_PARENT);
+ inode = dir->d_inode;
+ mutex_lock_nested(&inode->i_mutex, I_MUTEX_PARENT);
+ parent_sd = sysfs_resolve_for_remove(kobj, dir);
+ if (IS_ERR(parent_sd))
+  goto out_unlock;
  for (pos = &parent_sd->s_children; *pos; pos = &(*pos)->s_sibling) {
   sd = *pos;

@@ -298,11 +303,13 @@ 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 (!found)
   return -ENOENT;

+ sysfs_prune_shadow_sd(parent_sd);
  sysfs_drop_dentry(sd);
  sysfs_deactivate(sd);
  sysfs_put(sd);
diff --git a/fs/sysfs/symlink.c b/fs/sysfs/symlink.c
index ff605d3..ae2129c 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 (sd->s_type == SYSFS_SHADOW)
+   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 (sd->s_type == SYSFS_SHADOW)
+   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 (sd->s_type == SYSFS_SHADOW)
+   continue;
+
+  cur = strlen(sd->s_name);

   /* back up enough to print this bus id with '/' */
   length -= cur;
@@ -67,7 +77,7 @@ static int sysfs_add_link(struct sysfs_dirent * parent_sd, const char * name,
 int sysfs_create_link(struct kobject * kobj, struct kobject * target, const char * name)
 {
  struct dentry *dentry = NULL;
- struct sysfs_dirent *parent_sd = NULL;
+ struct sysfs_dirent *parent_sd;
  struct sysfs_dirent *target_sd = NULL;
  int error = -EEXIST;

@@ -81,7 +91,6 @@ int sysfs_create_link(struct kobject * kobj, struct kobject * target, const char

  if (!dentry)
   return -EFAULT;
- parent_sd = dentry->d_fsdata;
```

```
  /* target->dentry can go away beneath us but is protected with
   * kobj_sysfs_assoc_lock.  Fetch target_sd from it.
@@ -95,7 +104,10 @@ int sysfs_create_link(struct kobject * kobj, struct kobject * target, const
char
   return -ENOENT;

  mutex_lock(&dentry->d_inode->i_mutex);
- if (!sysfs_dirent_exist(dentry->d_fsdata, name))
+ parent_sd = sysfs_resolve_for_create(target, dentry);
+ if (IS_ERR(parent_sd))
+  error = PTR_ERR(parent_sd);
+ else if (!sysfs_dirent_exist(parent_sd, name))
   error = sysfs_add_link(parent_sd, name, target_sd);
  mutex_unlock(&dentry->d_inode->i_mutex);

@@ -114,7 +126,7 @@ 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 sysfs_dirent * parent_sd,
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index 6258462..1c4d20f 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -47,6 +47,10 @@ struct sysfs_dirent {
 extern struct vfsmount * sysfs_mount;
 extern struct kmem_cache *sysfs_dir_cachep;

+extern void sysfs_prune_shadow_sd(struct sysfs_dirent *sd);
+struct sysfs_dirent *sysfs_resolve_for_create(struct kobject *, struct dentry *);
+struct sysfs_dirent *sysfs_resolve_for_remove(struct kobject *, struct dentry *);
+
 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);
@@ -66,7 +70,7 @@ extern void sysfs_attach_dirent(struct sysfs_dirent *sd,
     struct dentry *dentry);

 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 **);
diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
index a3fa5b9..3db9b16 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -72,11 +72,17 @@ 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_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)
 #define SYSFS_COPY_NAME  (SYSFS_DIR | SYSFS_KOBJ_LINK)

@@ -129,6 +135,8 @@ 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 */
@@ -224,6 +232,12 @@ 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

---

Subject: [PATCH 3/4] sysfs: Implement sysfs_delete_link and sysfs_rename_link
Posted by ebiederm on Fri, 22 Jun 2007 07:37:30 GMT
View Forum Message <> Reply to Message

When removing a symlink sysfs_remove_link does not provide enough
information to figure out which shadow directory the symlink falls in.
So I need sysfs_delete_link which is passed the target of the symlink
to delete.

Further half the time when we are removing a symlink the code is
actually renaming the symlink but not doing so explicitly because we
don't have a symlink rename method. So I have added sysfs_rename_link
as well.

Both of these functions now have enough information to find a symlink
in a shadow directory. The only restriction is that they must be called
before the target kobject is renamed or deleted. If they are called
later I loose track of which tag the target kobject was marked with
and can no longer find the old symlink to remove it.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/symlink.c | 31 +++++++++++++++++++++++++++++++
 include/linux/sysfs.h | 18 ++++++++++++++++++
 2 files changed, 49 insertions(+), 0 deletions(-)

diff --git a/fs/sysfs/symlink.c b/fs/sysfs/symlink.c
index ae2129c..d483d6d 100644
--- a/fs/sysfs/symlink.c
+++ b/fs/sysfs/symlink.c
@@ -119,6 +119,21 @@ int sysfs_create_link(struct kobject * kobj, struct kobject * target, const
char

 /**
+ * sysfs_delete_link - remove symlink in object's directory.
+ * @kobj: object we're acting for.
+ * @targ: object we're pointing to.
+ * @name: name of the symlink to remove.
+ *

```
+ * Unlike sysfs_remove_link sysfs_delete_link has enough information
+ * to successfully delete symlinks in shadow directories.
+ */
+void sysfs_delete_link(struct kobject *kobj, struct kobject *targ,
+    const char *name)
+{
+ sysfs_hash_and_remove(targ, kobj->dentry,name);
+}
+
+/**
  * sysfs_remove_link - remove symlink in object's directory.
  * @kobj: object we're acting for.
  * @name: name of the symlink to remove.
@@ -129,6 +144,22 @@ void sysfs_remove_link(struct kobject * kobj, const char * name)
  sysfs_hash_and_remove(kobj, kobj->dentry,name);
 }

+/**
+ * sysfs_rename_link - rename symlink in object's directory.
+ * @kobj: object we're acting for.
+ * @targ: object we're pointing to.
+ * @old: previous name of the symlink.
+ * @new: new name of the symlink.
+ *
+ * A helper function for the common rename symlink idiom.
+ */
+int sysfs_rename_link(struct kobject *kobj, struct kobject *targ,
+    const char *old, const char *new)
+{
+ sysfs_delete_link(kobj, targ, old);
+ return sysfs_create_link(kobj, targ, new);
+}
+
 static int sysfs_get_target_path(struct sysfs_dirent * parent_sd,
      struct sysfs_dirent * target_sd, char *path)
 {
diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
index 3db9b16..c7f92b5 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -121,6 +121,13 @@ sysfs_create_link(struct kobject * kobj, struct kobject * target, const char
* n
 extern void
 sysfs_remove_link(struct kobject *, const char * name);

+extern int
+sysfs_rename_link(struct kobject *kobj, struct kobject *target,
+    const char *old_name, const char *new_name);
```

```
+
+extern void
+sysfs_delete_link(struct kobject *dir, struct kobject *targ, const char *name);
+
 int __must_check sysfs_create_bin_file(struct kobject *kobj,
      struct bin_attribute *attr);
 void sysfs_remove_bin_file(struct kobject *kobj, struct bin_attribute *attr);
@@ -196,6 +203,17 @@ static inline void sysfs_remove_link(struct kobject * k, const char *
name)
  ;
 }

+static inline int
+sysfs_rename_link(struct kobject * k, struct kobject *t,
+   const char *old_name, const char * new_name)
+{
+ return 0;
+}
+
+static inline void
+sysfs_delete_link(struct kobject *k, struct kobject *t, const char *name)
+{
+}

 static inline int sysfs_create_bin_file(struct kobject * k, struct bin_attribute * a)
 {
--
1.5.1.1.181.g2de0
```

Subject: PATCH 4/4] driver core: Implement shadow directory support for device classes.

Posted by ebiederm on Fri, 22 Jun 2007 07:39:06 GMT

View Forum Message <> Reply to Message

This patch enables shadowing on every class directory if struct class
has shadow_ops.

In addition device_del and device_rename were modified to use
sysfs_delete_link and sysfs_rename_link respectively to ensure when
these operations happen on devices whos classes have shadow operations
that they work properly.

```
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/base/class.c   |   30 ++++++++++++++++++++++++++----
 drivers/base/core.c    |   45 ++++++++++++++++++++++++++++++---------------
 include/linux/device.h |    2 ++
 3 files changed, 52 insertions(+), 25 deletions(-)

diff --git a/drivers/base/class.c b/drivers/base/class.c
index 4d22226..c981f75 100644
--- a/drivers/base/class.c
+++ b/drivers/base/class.c
@@ -134,6 +134,17 @@ static void remove_class_attrs(struct class * cls)
  }
 }

+static int class_setup_shadowing(struct class *cls)
+{
+ const struct shadow_dir_operations *shadow_ops;
+
+ shadow_ops = cls->shadow_ops;
+ if (!shadow_ops)
+  return 0;
+
+ return sysfs_enable_shadowing(&cls->subsys.kobj, shadow_ops);
+}
+
 int class_register(struct class * cls)
 {
  int error;
@@ -152,11 +163,22 @@ int class_register(struct class * cls)
  subsys_set_kset(cls, class_subsys);

  error = subsystem_register(&cls->subsys);
- if (!error) {
-  error = add_class_attrs(class_get(cls));
-  class_put(cls);
- }
+ if (error)
+  goto out;
+
+ error = class_setup_shadowing(cls);
+ if (error)
+  goto out_unregister;
+
+ error = add_class_attrs(cls);
+ if (error)
+  goto out_unregister;
+
```

```
+out:
  return error;
+out_unregister:
+ subsystem_unregister(&cls->subsys);
+ goto out;
 }

 void class_unregister(struct class * cls)
diff --git a/drivers/base/core.c b/drivers/base/core.c
index c7543bc..b1a5241 100644
--- a/drivers/base/core.c
+++ b/drivers/base/core.c
@@ -622,8 +622,14 @@ static struct kobject * get_device_parent(struct device *dev,
    return kobj;

   /* or create a new class-directory at the parent device */
-  return kobject_kset_add_dir(&dev->class->class_dirs,
+  kobj = kobject_kset_add_dir(&dev->class->class_dirs,
         parent_kobj, dev->class->name);
+
+  /* If we created a new class-directory setup shadowing */
+  if (kobj && dev->class->shadow_ops)
+   sysfs_enable_shadowing(kobj, dev->class->shadow_ops);
+
+  return kobj;
  }

  if (parent)
@@ -920,8 +926,8 @@ void device_del(struct device * dev)
   /* If this is not a "fake" compatible device, remove the
    * symlink from the class to the device. */
   if (dev->kobj.parent != &dev->class->subsys.kobj)
-   sysfs_remove_link(&dev->class->subsys.kobj,
-       dev->bus_id);
+   sysfs_delete_link(&dev->class->subsys.kobj,
+       &dev->kobj, dev->bus_id);
   if (parent) {
 #ifdef CONFIG_SYSFS_DEPRECATED
    char *class_name = make_class_name(dev->class->name,
@@ -1219,6 +1225,13 @@ int device_rename(struct device *dev, char *new_name)
  strlcpy(old_device_name, dev->bus_id, BUS_ID_SIZE);
  strlcpy(dev->bus_id, new_name, BUS_ID_SIZE);

+ if (dev->class && (dev->kobj.parent != &dev->class->subsys.kobj)) {
+  error = sysfs_rename_link(&dev->class->subsys.kobj,
+   &dev->kobj, old_device_name, new_name);
+  if (error)
+   goto out;
```

```
+ }
+
  error = kobject_rename(&dev->kobj, new_name);
  if (error) {
   strlcpy(dev->bus_id, old_device_name, BUS_ID_SIZE);
@@ -1227,27 +1240,17 @@ int device_rename(struct device *dev, char *new_name)

 #ifdef CONFIG_SYSFS_DEPRECATED
  if (old_class_name) {
+  error = -ENOMEM;
   new_class_name = make_class_name(dev->class->name, &dev->kobj);
-  if (new_class_name) {
-   error = sysfs_create_link(&dev->parent->kobj,
-         &dev->kobj, new_class_name);
-   if (error)
-    goto out;
-   sysfs_remove_link(&dev->parent->kobj, old_class_name);
-  }
- }
-#endif
+  if (!new_class_name)
+   goto out;

- if (dev->class) {
-  sysfs_remove_link(&dev->class->subsys.kobj, old_device_name);
-  error = sysfs_create_link(&dev->class->subsys.kobj, &dev->kobj,
-        dev->bus_id);
-  if (error) {
-   /* Uh... how to unravel this if restoring can fail? */
-   dev_err(dev, "%s: sysfs_create_symlink failed (%d)\n",
-    __FUNCTION__, error);
-  }
+  error = sysfs_rename_link(&dev->parent->kobj, &dev->kobj,
+        old_class_name, new_class_name);
+  if (error)
+   goto out;
  }
+#endif
 out:
  put_device(dev);

diff --git a/include/linux/device.h b/include/linux/device.h
index be2debe..918dfa3 100644
--- a/include/linux/device.h
+++ b/include/linux/device.h
@@ -200,6 +200,8 @@ struct class {

  int (*suspend)(struct device *, pm_message_t state);
```

```
  int (*resume)(struct device *);
+
+ const struct shadow_dir_operations *shadow_ops;
 };

 extern int __must_check class_register(struct class *);
--
1.5.1.1.181.g2de0
```

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

---

Subject: Re: [PATCH 1/4] sysfs: Remove first pass at shadow directory support
Posted by Benjamin Thery on Fri, 22 Jun 2007 14:42:01 GMT
View Forum Message <> Reply to Message

Eric,

Thanks a lot for the updated patchset.

I haven't look at the details yet, but I've already integrated it to
your network namespaces patchset and ported the whole thing to
2.6.22-rc4-mm2 today.
It is far easier to do when you don't have to think about the sysfs stuff :)

I only performed some basic testing on it so far as I'm home today, but
it already seems to work better than the patchset I ported to
2.6.21-mm2. It solves the issues I still had with sysfs.

Thanks again.

Regards,
Benjamin

Eric W. Biederman wrote:
> While shadow directories appear to be a good idea, the current scheme
> of controlling their creation and destruction outside of sysfs appears
> to be a locking and maintenance nightmare in the face of sysfs
> directories dynamically coming and going. Which can now occur for
> directories containing network devices when CONFIG_SYSFS_DEPRECATED is
> not set.
>
> This patch removes everything from the initial shadow directory
> support that allowed the shadow directory creation to be controlled at

> a higher level. So except for a few bits of sysfs_rename_dir
> everything from commit b592fcfe7f06c15ec11774b5be7ce0de3aa86e73 is now
> gone.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
> ---
> These patches are against 2.6.22-rc4-mm2 Hopefully that is new enough
> to catch all of the in flight sysfs patches.
>
> fs/sysfs/dir.c | 172 ++++++++++-----------------------------------
> fs/sysfs/group.c | 1 -
> fs/sysfs/inode.c | 10 ---
> fs/sysfs/mount.c | 2 +-
> fs/sysfs/sysfs.h | 6 --
> include/linux/kobject.h | 4 -
> include/linux/sysfs.h | 24 +------
> lib/kobject.c | 44 ++----------
> 8 files changed, 48 insertions(+), 215 deletions(-)
>
> diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
> index b4074ad..b1da4fc 100644
> --- a/fs/sysfs/dir.c
> +++ b/fs/sysfs/dir.c
> @@ -415,10 +415,9 @@ int sysfs_create_subdir(struct kobject * k, const char * n, struct dentry
** d)
> /**
>   * sysfs_create_dir - create a directory for an object.
>   * @kobj: object we're creating directory for.
> - * @shadow_parent: parent parent object.
>   */
>
> -int sysfs_create_dir(struct kobject * kobj, struct dentry *shadow_parent)
> +int sysfs_create_dir(struct kobject * kobj)
> {
>   struct dentry * dentry = NULL;
>   struct dentry * parent;
> @@ -426,9 +425,7 @@ int sysfs_create_dir(struct kobject * kobj, struct dentry
*shadow_parent)
>
>   BUG_ON(!kobj);
>
> - if (shadow_parent)
> - parent = shadow_parent;
> - else if (kobj->parent)
> + if (kobj->parent)
>   parent = kobj->parent->dentry;
>   else if (sysfs_mount && sysfs_mount->mnt_sb)
>   parent = sysfs_mount->mnt_sb->s_root;

```
> @@ -516,12 +513,26 @@ void sysfs_remove_subdir(struct dentry * d)
>  }
>
>
> -static void __sysfs_remove_dir(struct dentry *dentry)
> +/**
> + * sysfs_remove_dir - remove an object's directory.
> + * @kobj: object.
> + *
> + * The only thing special about this is that we remove any files in
> + * the directory before we remove the directory, and we've inlined
> + * what used to be sysfs_rmdir() below, instead of calling separately.
> + */
> +
> +void sysfs_remove_dir(struct kobject * kobj)
>  {
> + struct dentry *dentry = kobj->dentry;
>   struct sysfs_dirent *removed = NULL;
>   struct sysfs_dirent *parent_sd;
>   struct sysfs_dirent **pos;
>
> + spin_lock(&kobj_sysfs_assoc_lock);
> + kobj->dentry = NULL;
> + spin_unlock(&kobj_sysfs_assoc_lock);
> +
>   if (!dentry)
>    return;
>
> @@ -555,55 +566,35 @@ static void __sysfs_remove_dir(struct dentry *dentry)
>   remove_dir(dentry);
>  }
>
> -/**
> - * sysfs_remove_dir - remove an object's directory.
> - * @kobj: object.
> - *
> - * The only thing special about this is that we remove any files in
> - * the directory before we remove the directory, and we've inlined
> - * what used to be sysfs_rmdir() below, instead of calling separately.
> - */
> -
> -void sysfs_remove_dir(struct kobject * kobj)
> -{
> - struct dentry *d = kobj->dentry;
> -
> - spin_lock(&kobj_sysfs_assoc_lock);
> - kobj->dentry = NULL;
> - spin_unlock(&kobj_sysfs_assoc_lock);
```

```
> -
> - __sysfs_remove_dir(d);
> -}
> -
> -int sysfs_rename_dir(struct kobject * kobj, struct dentry *new_parent,
> -       const char *new_name)
> +int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
>  {
> - struct sysfs_dirent *sd = kobj->dentry->d_fsdata;
> - struct sysfs_dirent *parent_sd = new_parent->d_fsdata;
> - struct dentry *new_dentry;
> + struct sysfs_dirent *sd;
> + struct sysfs_dirent *parent_sd;
> + struct inode *inode;
> + struct dentry *new_dentry, *parent;
>   char *dup_name;
>   int error;
>
> - if (!new_parent)
> -  return -EFAULT;
> + if (!kobj->parent)
> +  return -EINVAL;
> +
> + parent = dget(kobj->dentry->d_parent);
> + inode = parent->d_inode;
> +
> + sd = kobj->dentry->d_fsdata;
>
>   down_write(&sysfs_rename_sem);
> - mutex_lock(&new_parent->d_inode->i_mutex);
> + mutex_lock(&inode->i_mutex);
>
> - new_dentry = lookup_one_len(new_name, new_parent, strlen(new_name));
> + parent_sd = parent->d_fsdata;
> + new_dentry = lookup_one_len(new_name, parent_sd->s_dentry, strlen(new_name));
>   if (IS_ERR(new_dentry)) {
>    error = PTR_ERR(new_dentry);
>    goto out_unlock;
>   }
>
> - /* By allowing two different directories with the same
> -  * d_parent we allow this routine to move between different
> -  * shadows of the same directory
> -  */
>   error = -EINVAL;
> - if (kobj->dentry->d_parent->d_inode != new_parent->d_inode ||
> -     new_dentry->d_parent->d_inode != new_parent->d_inode ||
> -     new_dentry == kobj->dentry)
```

```
> + if (new_dentry == kobj->dentry)
>     goto out_dput;
>
>   error = -EEXIST;
> @@ -643,8 +634,11 @@ int sysfs_rename_dir(struct kobject * kobj, struct dentry *new_parent,
>   out_dput:
>   dput(new_dentry);
>   out_unlock:
> - mutex_unlock(&new_parent->d_inode->i_mutex);
> + mutex_unlock(&inode->i_mutex);
>   up_write(&sysfs_rename_sem);
> +
> + dput(parent);
> +
>   return error;
>  }
>
> @@ -837,98 +831,6 @@ static loff_t sysfs_dir_lseek(struct file * file, loff_t offset, int origin)
>   return offset;
>  }
>
> -
> -/**
> - * sysfs_make_shadowed_dir - Setup so a directory can be shadowed
> - * @kobj: object we're creating shadow of.
> - */
> -
> -int sysfs_make_shadowed_dir(struct kobject *kobj,
> - void * (*follow_link)(struct dentry *, struct nameidata *))
> -{
> - struct inode *inode;
> - struct inode_operations *i_op;
> -
> - inode = kobj->dentry->d_inode;
> - if (inode->i_op != &sysfs_dir_inode_operations)
> -  return -EINVAL;
> -
> - i_op = kmalloc(sizeof(*i_op), GFP_KERNEL);
> - if (!i_op)
> -  return -ENOMEM;
> -
> - memcpy(i_op, &sysfs_dir_inode_operations, sizeof(*i_op));
> - i_op->follow_link = follow_link;
> -
> - /* Locking of inode->i_op?
> -  * Since setting i_op is a single word write and they
> -  * are atomic we should be ok here.
> -  */
```

> - inode->i_op = i_op;
> - return 0;
> -}
> -
> -/**
> - * sysfs_create_shadow_dir - create a shadow directory for an object.
> - * @kobj: object we're creating directory for.
> - *
> - * sysfs_make_shadowed_dir must already have been called on this
> - * directory.
> - */
> -
> -struct dentry *sysfs_create_shadow_dir(struct kobject *kobj)
> -{
> - struct dentry *dir = kobj->dentry;
> - struct inode *inode = dir->d_inode;
> - struct dentry *parent = dir->d_parent;
> - struct sysfs_dirent *parent_sd = parent->d_fsdata;
> - struct dentry *shadow;
> - struct sysfs_dirent *sd;
> -
> - shadow = ERR_PTR(-EINVAL);
> - if (!sysfs_is_shadowed_inode(inode))
> - goto out;
> -
> - shadow = d_alloc(parent, &dir->d_name);
> - if (!shadow)
> - goto nomem;
> -
> - sd = sysfs_new_dirent("_SHADOW_", inode->i_mode, SYSFS_DIR);
> - if (!sd)
> - goto nomem;
> - sd->s_elem.dir.kobj = kobj;
> - /* point to parent_sd but don't attach to it */
> - sd->s_parent = sysfs_get(parent_sd);
> - sysfs_attach_dirent(sd, NULL, shadow);
> -
> - d_instantiate(shadow, igrab(inode));
> - inc_nlink(inode);
> - inc_nlink(parent->d_inode);
> -
> - dget(shadow); /* Extra count - pin the dentry in core */
> -
> -out:
> - return shadow;
> -nomem:
> - dput(shadow);
> - shadow = ERR_PTR(-ENOMEM);

> - goto out;
> -}
> -
> -/**
> - * sysfs_remove_shadow_dir - remove an object's directory.
> - * @shadow: dentry of shadow directory
> - *
> - * The only thing special about this is that we remove any files in
> - * the directory before we remove the directory, and we've inlined
> - * what used to be sysfs_rmdir() below, instead of calling separately.
> - */
> -
> -void sysfs_remove_shadow_dir(struct dentry *shadow)
> -{
> - __sysfs_remove_dir(shadow);
> -}
> -
> const struct file_operations sysfs_dir_operations = {
>   .open = sysfs_dir_open,
>   .release = sysfs_dir_close,
> diff --git a/fs/sysfs/group.c b/fs/sysfs/group.c
> index 52eed2a..82e0f55 100644
> --- a/fs/sysfs/group.c
> +++ b/fs/sysfs/group.c
> @@ -13,7 +13,6 @@
> #include <linux/dcache.h>
> #include <linux/namei.h>
> #include <linux/err.h>
> -#include <linux/fs.h>
> #include <asm/semaphore.h>
> #include "sysfs.h"
>
> diff --git a/fs/sysfs/inode.c b/fs/sysfs/inode.c
> index b1a4527..8181c49 100644
> --- a/fs/sysfs/inode.c
> +++ b/fs/sysfs/inode.c
> @@ -34,16 +34,6 @@ static const struct inode_operations sysfs_inode_operations ={
>   .setattr = sysfs_setattr,
> };
>
> -void sysfs_delete_inode(struct inode *inode)
> -{
> - /* Free the shadowed directory inode operations */
> - if (sysfs_is_shadowed_inode(inode)) {
> - kfree(inode->i_op);
> - inode->i_op = NULL;
> - }
> - return generic_delete_inode(inode);

> -}
> -
> int sysfs_setattr(struct dentry * dentry, struct iattr * iattr)
> {
>   struct inode * inode = dentry->d_inode;
> diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
> index 4be9593..c8126ab 100644
> --- a/fs/sysfs/mount.c
> +++ b/fs/sysfs/mount.c
> @@ -21,7 +21,7 @@ struct kmem_cache *sysfs_dir_cachep;
>
> static const struct super_operations sysfs_ops = {
>   .statfs = simple_statfs,
> - .drop_inode = sysfs_delete_inode,
> + .drop_inode = generic_delete_inode,
> };
>
> static struct sysfs_dirent sysfs_root = {
> diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
> index 6f8aaf3..6258462 100644
> --- a/fs/sysfs/sysfs.h
> +++ b/fs/sysfs/sysfs.h
> @@ -53,7 +53,6 @@ extern struct sysfs_dirent *sysfs_get_active_two(struct sysfs_dirent *sd);
> extern void sysfs_put_active_two(struct sysfs_dirent *sd);
> extern void sysfs_deactivate(struct sysfs_dirent *sd);
>
> -extern void sysfs_delete_inode(struct inode *inode);
> extern void sysfs_init_inode(struct sysfs_dirent *sd, struct inode *inode);
> extern struct inode * sysfs_get_inode(struct sysfs_dirent *sd);
> extern void sysfs_instantiate(struct dentry *dentry, struct inode *inode);
> @@ -100,8 +99,3 @@ static inline void sysfs_put(struct sysfs_dirent * sd)
>   if (sd && atomic_dec_and_test(&sd->s_count))
>   release_sysfs_dirent(sd);
> }
> -
> -static inline int sysfs_is_shadowed_inode(struct inode *inode)
> -{
> - return S_ISDIR(inode->i_mode) && inode->i_op->follow_link;
> -}
> diff --git a/include/linux/kobject.h b/include/linux/kobject.h
> index c288e41..4fa5b88 100644
> --- a/include/linux/kobject.h
> +++ b/include/linux/kobject.h
> @@ -71,13 +71,9 @@ extern void kobject_init(struct kobject *);
> extern void kobject_cleanup(struct kobject *);
>
> extern int __must_check kobject_add(struct kobject *);
> -extern int __must_check kobject_shadow_add(struct kobject *, struct dentry *);

> extern void kobject_del(struct kobject *);
>
> extern int __must_check kobject_rename(struct kobject *, const char *new_name);
> -extern int __must_check kobject_shadow_rename(struct kobject *kobj,
> -   struct dentry *new_parent,
> -   const char *new_name);
> extern int __must_check kobject_move(struct kobject *, struct kobject *);
>
> extern int __must_check kobject_register(struct kobject *);
> diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
> index 161e19a..a3fa5b9 100644
> --- a/include/linux/sysfs.h
> +++ b/include/linux/sysfs.h
> @@ -17,8 +17,6 @@
>
> struct kobject;
> struct module;
> -struct nameidata;
> -struct dentry;
>
> /* FIXME
>   * The *owner field is no longer used, but leave around
> @@ -88,13 +86,13 @@ extern int sysfs_schedule_callback(struct kobject *kobj,
>   void (*func)(void *), void *data, struct module *owner);
>
> extern int __must_check
> -sysfs_create_dir(struct kobject *, struct dentry *);
> +sysfs_create_dir(struct kobject *);
>
> extern void
> sysfs_remove_dir(struct kobject *);
>
> extern int __must_check
> -sysfs_rename_dir(struct kobject *, struct dentry *, const char *new_name);
> +sysfs_rename_dir(struct kobject *, const char *new_name);
>
> extern int __must_check
> sysfs_move_dir(struct kobject *, struct kobject *);
> @@ -131,12 +129,6 @@ void sysfs_remove_file_from_group(struct kobject *kobj,
>
> void sysfs_notify(struct kobject * k, char *dir, char *attr);
>
> -
> -extern int sysfs_make_shadowed_dir(struct kobject *kobj,
> - void * (*follow_link)(struct dentry *, struct nameidata *));
> -extern struct dentry *sysfs_create_shadow_dir(struct kobject *kobj);
> -extern void sysfs_remove_shadow_dir(struct dentry *dir);
> -

> extern int __must_check sysfs_init(void);
>
> #else /* CONFIG_SYSFS */
> @@ -147,7 +139,7 @@ static inline int sysfs_schedule_callback(struct kobject *kobj,
>   return -ENOSYS;
> }
>
> -static inline int sysfs_create_dir(struct kobject * k, struct dentry *shadow)
> +static inline int sysfs_create_dir(struct kobject * k)
> {
>   return 0;
> }
> @@ -157,9 +149,7 @@ static inline void sysfs_remove_dir(struct kobject * k)
>   ;
> }
>
> -static inline int sysfs_rename_dir(struct kobject * k,
> -   struct dentry *new_parent,
> -   const char *new_name)
> +static inline int sysfs_rename_dir(struct kobject * k, const char *new_name)
> {
>   return 0;
> }
> @@ -234,12 +224,6 @@ static inline void sysfs_notify(struct kobject * k, char *dir, char *attr)
> {
> }
>
> -static inline int sysfs_make_shadowed_dir(struct kobject *kobj,
> - void * (*follow_link)(struct dentry *, struct nameidata *))
> -{
> - return 0;
> -}
> -
> static inline int __must_check sysfs_init(void)
> {
>   return 0;
> diff --git a/lib/kobject.c b/lib/kobject.c
> index cb40a00..738d76f 100644
> --- a/lib/kobject.c
> +++ b/lib/kobject.c
> @@ -46,11 +46,11 @@ static int populate_dir(struct kobject * kobj)
>   return error;
> }
>
> -static int create_dir(struct kobject * kobj, struct dentry *shadow_parent)
> +static int create_dir(struct kobject * kobj)
> {
>   int error = 0;

>   if (kobject_name(kobj)) {
> - error = sysfs_create_dir(kobj, shadow_parent);
> + error = sysfs_create_dir(kobj);
>   if (!error) {
>   if ((error = populate_dir(kobj)))
>   sysfs_remove_dir(kobj);
> @@ -201,12 +201,11 @@ static void unlink(struct kobject * kobj)
> }
>
> /**
> - * kobject_shadow_add - add an object to the hierarchy.
> + * kobject_add - add an object to the hierarchy.
>   * @kobj: object.
> - * @shadow_parent: sysfs directory to add to.
>   */
>
> -int kobject_shadow_add(struct kobject * kobj, struct dentry *shadow_parent)
> +int kobject_add(struct kobject * kobj)
> {
>   int error = 0;
>   struct kobject * parent;
> @@ -238,7 +237,7 @@ int kobject_shadow_add(struct kobject * kobj, struct dentry
*shadow_parent)
>   kobj->parent = parent;
>   }
>
> - error = create_dir(kobj, shadow_parent);
> + error = create_dir(kobj);
>   if (error) {
>   /* unlink does the kobject_put() for us */
>   unlink(kobj);
> @@ -260,16 +259,6 @@ int kobject_shadow_add(struct kobject * kobj, struct dentry
*shadow_parent)
> }
>
> /**
> - * kobject_add - add an object to the hierarchy.
> - * @kobj: object.
> - */
> -int kobject_add(struct kobject * kobj)
> -{
> - return kobject_shadow_add(kobj, NULL);
> -}
> -
> -
> -/**
>   * kobject_register - initialize and add an object.
>   * @kobj: object in question.

> */
> @@ -382,7 +371,7 @@ int kobject_rename(struct kobject * kobj, const char *new_name)
> /* Note : if we want to send the new name alone, not the full path,
>   * we could probably use kobject_name(kobj); */
>
> - error = sysfs_rename_dir(kobj, kobj->parent->dentry, new_name);
> + error = sysfs_rename_dir(kobj, new_name);
>
> /* This function is mostly/only used for network interface.
>   * Some hotplug package track interfaces by their name and
> @@ -399,27 +388,6 @@ out:
> }
>
> /**
> - * kobject_rename - change the name of an object
> - * @kobj: object in question.
> - * @new_parent: object's new parent
> - * @new_name: object's new name
> - */
> -
> -int kobject_shadow_rename(struct kobject * kobj, struct dentry *new_parent,
> -   const char *new_name)
> -{
> - int error = 0;
> -
> - kobj = kobject_get(kobj);
> - if (!kobj)
> - return -EINVAL;
> - error = sysfs_rename_dir(kobj, new_parent, new_name);
> - kobject_put(kobj);
> -
> - return error;
> -}
> -
> -/**
> * kobject_move - move object to another parent
> * @kobj: object in question.
> * @new_parent: object's new parent (can be NULL)

_____

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

---

Subject: Re: [PATCH 1/4] sysfs: Remove first pass at shadow directory support
Posted by Greg KH on Mon, 25 Jun 2007 21:23:39 GMT

On Fri, Jun 22, 2007 at 01:33:42AM -0600, Eric W. Biederman wrote:
>
> While shadow directories appear to be a good idea, the current scheme
> of controlling their creation and destruction outside of sysfs appears
> to be a locking and maintenance nightmare in the face of sysfs
> directories dynamically coming and going. Which can now occur for
> directories containing network devices when CONFIG_SYSFS_DEPRECATED is
> not set.
>
> This patch removes everything from the initial shadow directory
> support that allowed the shadow directory creation to be controlled at
> a higher level. So except for a few bits of sysfs_rename_dir
> everything from commit b592fcfe7f06c15ec11774b5be7ce0de3aa86e73 is now
> gone.
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
> ---
> These patches are against 2.6.22-rc4-mm2 Hopefully that is new enough
> to catch all of the in flight sysfs patches.

Ick, no, it isn't and doesn't apply at all :(

Can you try the next -mm release?

thanks,

greg k-h

Subject: Re: [PATCH 1/4] sysfs: Remove first pass at shadow directory support
Posted by ebiederm on Tue, 26 Jun 2007 00:50:19 GMT

Greg KH <greg@kroah.com> writes:

> Ick, no, it isn't and doesn't apply at all :(

Groan. I wonder what changed this time...

> Can you try the next -mm release?

Ok. As soon as I get back from OLS, I will rebase against

whatever is current.

Eric

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

---

Subject: [PATCH 1/4] sysfs: Remove first pass at shadow directory support
Posted by ebiederm on Thu, 19 Jul 2007 04:43:46 GMT
View Forum Message <> Reply to Message

While shadow directories appear to be a good idea, the current scheme
of controlling their creation and destruction outside of sysfs appears
to be a locking and maintenance nightmare in the face of sysfs
directories dynamically coming and going. Which can now occur for
directories containing network devices when CONFIG_SYSFS_DEPRECATED is
not set.

This patch removes everything from the initial shadow directory support
that allowed the shadow directory creation to be controlled at a higher
level. So except for a few bits of sysfs_rename_dir everything from
commit b592fcfe7f06c15ec11774b5be7ce0de3aa86e73 is now gone.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

This patchset is against 589f1e81bde732dd0b1bc5d01b6bddd4bcb4527b
in Linus's tree. I took a quick skim through your tree and all
of the sysfs patches appear to be merged.

Sorry for taking so long getting this done but the locking
changes in the last round for Tejun were significant. Plus
there was some additional work needed to actually test the
shadow directory support.

```
 fs/sysfs/dir.c          |  167 ++++++----------------------------------------
 fs/sysfs/group.c        |    1 -
 fs/sysfs/inode.c        |   10 ---
 fs/sysfs/mount.c        |    2 +-
 fs/sysfs/sysfs.h        |    6 --
 include/linux/kobject.h |    5 --
 include/linux/sysfs.h   |   27 +-------
 lib/kobject.c           |   44 ++-----------
 8 files changed, 33 insertions(+), 229 deletions(-)
```

diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
index 048e605..f88130c 100644

```
--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -569,9 +569,6 @@ static void sysfs_drop_dentry(struct sysfs_dirent *sd)
  spin_unlock(&dcache_lock);
  spin_unlock(&sysfs_assoc_lock);

- /* dentries for shadowed inodes are pinned, unpin */
- if (dentry && sysfs_is_shadowed_inode(dentry->d_inode))
-  dput(dentry);
  dput(dentry);

  /* adjust nlink and update timestamp */
@@ -723,19 +720,15 @@ int sysfs_create_subdir(struct kobject *kobj, const char *name,
 /**
  * sysfs_create_dir - create a directory for an object.
  * @kobj:  object we're creating directory for.
- * @shadow_parent: parent object.
  */
-int sysfs_create_dir(struct kobject *kobj,
-       struct sysfs_dirent *shadow_parent_sd)
+int sysfs_create_dir(struct kobject * kobj)
 {
  struct sysfs_dirent *parent_sd, *sd;
  int error = 0;

  BUG_ON(!kobj);

- if (shadow_parent_sd)
-  parent_sd = shadow_parent_sd;
- else if (kobj->parent)
+ if (kobj->parent)
   parent_sd = kobj->parent->sd;
  else if (sysfs_mount && sysfs_mount->mnt_sb)
   parent_sd = sysfs_mount->mnt_sb->s_root->d_fsdata;
@@ -887,45 +880,44 @@ void sysfs_remove_dir(struct kobject * kobj)
  __sysfs_remove_dir(sd);
 }

-int sysfs_rename_dir(struct kobject *kobj, struct sysfs_dirent *new_parent_sd,
-       const char *new_name)
+int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
 {
- struct sysfs_dirent *sd = kobj->sd;
- struct dentry *new_parent = NULL;
+ 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;
  int error;

+ if (!kobj->parent)
+  return -EINVAL;
+
  /* get dentries */
+ sd = kobj->sd;
  old_dentry = sysfs_get_dentry(sd);
  if (IS_ERR(old_dentry)) {
   error = PTR_ERR(old_dentry);
   goto out_dput;
  }

- new_parent = sysfs_get_dentry(new_parent_sd);
- if (IS_ERR(new_parent)) {
-  error = PTR_ERR(new_parent);
+ parent_sd = kobj->parent->sd;
+ parent = sysfs_get_dentry(parent_sd);
+ if (IS_ERR(parent)) {
+  error = PTR_ERR(parent);
   goto out_dput;
  }

- /* lock new_parent and get dentry for new name */
- mutex_lock(&new_parent->d_inode->i_mutex);
+ /* lock parent and get dentry for new name */
+ mutex_lock(&parent->d_inode->i_mutex);

- new_dentry = lookup_one_len(new_name, new_parent, strlen(new_name));
+ new_dentry = lookup_one_len(new_name, parent, strlen(new_name));
  if (IS_ERR(new_dentry)) {
   error = PTR_ERR(new_dentry);
   goto out_unlock;
  }

- /* By allowing two different directories with the same
-  * d_parent we allow this routine to move between different
-  * shadows of the same directory
-  */
  error = -EINVAL;
- if (old_dentry->d_parent->d_inode != new_parent->d_inode ||
-     new_dentry->d_parent->d_inode != new_parent->d_inode ||
-     old_dentry == new_dentry)
+ if (old_dentry == new_dentry)
   goto out_unlock;

  error = -EEXIST;
```

```
@@ -952,9 +944,9 @@ int sysfs_rename_dir(struct kobject *kobj, struct sysfs_dirent
*new_parent_sd,
  mutex_lock(&sysfs_mutex);

  sysfs_unlink_sibling(sd);
- sysfs_get(new_parent_sd);
+ sysfs_get(parent_sd);
  sysfs_put(sd->s_parent);
- sd->s_parent = new_parent_sd;
+ sd->s_parent = parent_sd;
  sysfs_link_sibling(sd);

  mutex_unlock(&sysfs_mutex);
@@ -965,10 +957,10 @@ int sysfs_rename_dir(struct kobject *kobj, struct sysfs_dirent
*new_parent_sd,
  out_drop:
  d_drop(new_dentry);
  out_unlock:
- mutex_unlock(&new_parent->d_inode->i_mutex);
+ mutex_unlock(&parent->d_inode->i_mutex);
  out_dput:
  kfree(dup_name);
- dput(new_parent);
+ dput(parent);
  dput(old_dentry);
  dput(new_dentry);
  return error;
@@ -1189,121 +1181,6 @@ static loff_t sysfs_dir_lseek(struct file * file, loff_t offset, int origin)
  return offset;
 }

-
-/**
- * sysfs_make_shadowed_dir - Setup so a directory can be shadowed
- * @kobj: object we're creating shadow of.
- */
-
-int sysfs_make_shadowed_dir(struct kobject *kobj,
- void * (*follow_link)(struct dentry *, struct nameidata *))
-{
- struct dentry *dentry;
- struct inode *inode;
- struct inode_operations *i_op;
-
- /* get dentry for @kobj->sd, dentry of a shadowed dir is pinned */
- dentry = sysfs_get_dentry(kobj->sd);
- if (IS_ERR(dentry))
-  return PTR_ERR(dentry);
```

```
-
- inode = dentry->d_inode;
- if (inode->i_op != &sysfs_dir_inode_operations) {
-  dput(dentry);
-  return -EINVAL;
- }
-
- i_op = kmalloc(sizeof(*i_op), GFP_KERNEL);
- if (!i_op)
-  return -ENOMEM;
-
- memcpy(i_op, &sysfs_dir_inode_operations, sizeof(*i_op));
- i_op->follow_link = follow_link;
-
- /* Locking of inode->i_op?
-  * Since setting i_op is a single word write and they
-  * are atomic we should be ok here.
-  */
- inode->i_op = i_op;
- return 0;
-}
-
-/**
- * sysfs_create_shadow_dir - create a shadow directory for an object.
- * @kobj: object we're creating directory for.
- *
- * sysfs_make_shadowed_dir must already have been called on this
- * directory.
- */
-
-struct sysfs_dirent *sysfs_create_shadow_dir(struct kobject *kobj)
-{
- struct sysfs_dirent *parent_sd = kobj->sd->s_parent;
- struct dentry *dir, *parent, *shadow;
- struct inode *inode;
- struct sysfs_dirent *sd;
- struct sysfs_addrm_cxt acxt;
-
- dir = sysfs_get_dentry(kobj->sd);
- if (IS_ERR(dir)) {
-  sd = (void *)dir;
-  goto out;
- }
- parent = dir->d_parent;
-
- inode = dir->d_inode;
- sd = ERR_PTR(-EINVAL);
- if (!sysfs_is_shadowed_inode(inode))
```

```
-  goto out_dput;
-
- shadow = d_alloc(parent, &dir->d_name);
- if (!shadow)
-  goto nomem;
-
- sd = sysfs_new_dirent("_SHADOW_", inode->i_mode, SYSFS_DIR);
- if (!sd)
-  goto nomem;
- sd->s_elem.dir.kobj = kobj;
-
- sysfs_addrm_start(&acxt, parent_sd);
-
- /* add but don't link into children list */
- sysfs_add_one(&acxt, sd);
-
- /* attach and instantiate dentry */
- sysfs_attach_dentry(sd, shadow);
- d_instantiate(shadow, igrab(inode));
- inc_nlink(inode); /* tj: synchronization? */
-
- sysfs_addrm_finish(&acxt);
-
- dget(shadow);  /* Extra count - pin the dentry in core */
-
- goto out_dput;
-
- nomem:
- dput(shadow);
- sd = ERR_PTR(-ENOMEM);
- out_dput:
- dput(dir);
- out:
- return sd;
-}
-
-/**
- * sysfs_remove_shadow_dir - remove an object's directory.
- * @shadow_sd: sysfs_dirent of shadow directory
- *
- * The only thing special about this is that we remove any files in
- * the directory before we remove the directory, and we've inlined
- * what used to be sysfs_rmdir() below, instead of calling separately.
- */
-
-void sysfs_remove_shadow_dir(struct sysfs_dirent *shadow_sd)
-{
- __sysfs_remove_dir(shadow_sd);
```

```
-}
-
 const struct file_operations sysfs_dir_operations = {
  .open  = sysfs_dir_open,
  .release = sysfs_dir_close,
diff --git a/fs/sysfs/group.c b/fs/sysfs/group.c
index f318b73..4606f7c 100644
--- a/fs/sysfs/group.c
+++ b/fs/sysfs/group.c
@@ -13,7 +13,6 @@
 #include <linux/dcache.h>
 #include <linux/namei.h>
 #include <linux/err.h>
-#include <linux/fs.h>
 #include <asm/semaphore.h>
 #include "sysfs.h"

diff --git a/fs/sysfs/inode.c b/fs/sysfs/inode.c
index 10d1b52..9671164 100644
--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -34,16 +34,6 @@ static const struct inode_operations sysfs_inode_operations ={
  .setattr = sysfs_setattr,
 };

-void sysfs_delete_inode(struct inode *inode)
-{
- /* Free the shadowed directory inode operations */
- if (sysfs_is_shadowed_inode(inode)) {
-  kfree(inode->i_op);
-  inode->i_op = NULL;
- }
- return generic_delete_inode(inode);
-}
-
 int sysfs_setattr(struct dentry * dentry, struct iattr * iattr)
 {
  struct inode * inode = dentry->d_inode;
diff --git a/fs/sysfs/mount.c b/fs/sysfs/mount.c
index 60714d0..e2073e6 100644
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -21,7 +21,7 @@ struct kmem_cache *sysfs_dir_cachep;

 static const struct super_operations sysfs_ops = {
  .statfs  = simple_statfs,
- .drop_inode = sysfs_delete_inode,
+ .drop_inode = generic_delete_inode,
```

```
 };

 struct sysfs_dirent sysfs_root = {
diff --git a/fs/sysfs/sysfs.h b/fs/sysfs/sysfs.h
index 6b8c8d7..b55e510 100644
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -70,7 +70,6 @@ extern void sysfs_remove_one(struct sysfs_addrm_cxt *acxt,
         struct sysfs_dirent *sd);
 extern int sysfs_addrm_finish(struct sysfs_addrm_cxt *acxt);

-extern void sysfs_delete_inode(struct inode *inode);
 extern struct inode * sysfs_get_inode(struct sysfs_dirent *sd);
 extern void sysfs_instantiate(struct dentry *dentry, struct inode *inode);

@@ -121,8 +120,3 @@ static inline void sysfs_put(struct sysfs_dirent * sd)
   if (sd && atomic_dec_and_test(&sd->s_count))
     release_sysfs_dirent(sd);
 }
-
-static inline int sysfs_is_shadowed_inode(struct inode *inode)
-{
- return S_ISDIR(inode->i_mode) && inode->i_op->follow_link;
-}
diff --git a/include/linux/kobject.h b/include/linux/kobject.h
index aa2fe22..7108a3e 100644
--- a/include/linux/kobject.h
+++ b/include/linux/kobject.h
@@ -80,14 +80,9 @@ extern void kobject_init(struct kobject *);
 extern void kobject_cleanup(struct kobject *);

 extern int __must_check kobject_add(struct kobject *);
-extern int __must_check kobject_shadow_add(struct kobject *kobj,
-        struct sysfs_dirent *shadow_parent);
 extern void kobject_del(struct kobject *);

 extern int __must_check kobject_rename(struct kobject *, const char *new_name);
-extern int __must_check kobject_shadow_rename(struct kobject *kobj,
-          struct sysfs_dirent *new_parent,
-          const char *new_name);
 extern int __must_check kobject_move(struct kobject *, struct kobject *);

 extern int __must_check kobject_register(struct kobject *);
diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
index be8228e..c16e4c5 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -17,9 +17,6 @@
```

```
 struct kobject;
 struct module;
-struct nameidata;
-struct dentry;
-struct sysfs_dirent;

 /* FIXME
  * The *owner field is no longer used, but leave around
@@ -94,14 +91,13 @@ extern int sysfs_schedule_callback(struct kobject *kobj,
   void (*func)(void *), void *data, struct module *owner);

 extern int __must_check
-sysfs_create_dir(struct kobject *kobj, struct sysfs_dirent *shadow_parent_sd);
+sysfs_create_dir(struct kobject *);

 extern void
 sysfs_remove_dir(struct kobject *);

 extern int __must_check
-sysfs_rename_dir(struct kobject *kobj, struct sysfs_dirent *new_parent_sd,
-   const char *new_name);
+sysfs_rename_dir(struct kobject *kobj, const char *new_name);

 extern int __must_check
 sysfs_move_dir(struct kobject *, struct kobject *);
@@ -138,12 +134,6 @@ void sysfs_remove_file_from_group(struct kobject *kobj,

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

-
-extern int sysfs_make_shadowed_dir(struct kobject *kobj,
- void * (*follow_link)(struct dentry *, struct nameidata *));
-extern struct sysfs_dirent *sysfs_create_shadow_dir(struct kobject *kobj);
-extern void sysfs_remove_shadow_dir(struct sysfs_dirent *shadow_sd);
-
 extern int __must_check sysfs_init(void);

 #else /* CONFIG_SYSFS */
@@ -154,8 +144,7 @@ static inline int sysfs_schedule_callback(struct kobject *kobj,
  return -ENOSYS;
 }

-static inline int sysfs_create_dir(struct kobject *kobj,
-       struct sysfs_dirent *shadow_parent_sd)
+static inline int sysfs_create_dir(struct kobject * kobj)
 {
  return 0;
```

```
 }
@@ -165,9 +154,7 @@ static inline void sysfs_remove_dir(struct kobject * k)
  ;
 }

-static inline int sysfs_rename_dir(struct kobject *kobj,
-       struct sysfs_dirent *new_parent_sd,
-       const char *new_name)
+static inline int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
 {
  return 0;
 }
@@ -242,12 +229,6 @@ static inline void sysfs_notify(struct kobject * k, char *dir, char *attr)
 {
 }

-static inline int sysfs_make_shadowed_dir(struct kobject *kobj,
- void * (*follow_link)(struct dentry *, struct nameidata *))
-{
- return 0;
-}
-
 static inline int __must_check sysfs_init(void)
 {
  return 0;
diff --git a/lib/kobject.c b/lib/kobject.c
index 4b08e0f..2fc9fc6 100644
--- a/lib/kobject.c
+++ b/lib/kobject.c
@@ -44,11 +44,11 @@ static int populate_dir(struct kobject * kobj)
  return error;
 }

-static int create_dir(struct kobject *kobj, struct sysfs_dirent *shadow_parent)
+static int create_dir(struct kobject * kobj)
 {
  int error = 0;
  if (kobject_name(kobj)) {
-  error = sysfs_create_dir(kobj, shadow_parent);
+  error = sysfs_create_dir(kobj);
   if (!error) {
    if ((error = populate_dir(kobj)))
     sysfs_remove_dir(kobj);
@@ -157,12 +157,11 @@ static void unlink(struct kobject * kobj)
 }

 /**
- * kobject_shadow_add - add an object to the hierarchy.
```

```
+ * kobject_add - add an object to the hierarchy.
  * @kobj: object.
- * @shadow_parent: sysfs directory to add to.
  */

-int kobject_shadow_add(struct kobject *kobj, struct sysfs_dirent *shadow_parent)
+int kobject_add(struct kobject * kobj)
 {
  int error = 0;
  struct kobject * parent;
@@ -194,7 +193,7 @@ int kobject_shadow_add(struct kobject *kobj, struct sysfs_dirent
*shadow_parent)
   kobj->parent = parent;
  }

- error = create_dir(kobj, shadow_parent);
+ error = create_dir(kobj);
  if (error) {
   /* unlink does the kobject_put() for us */
   unlink(kobj);
@@ -216,16 +215,6 @@ int kobject_shadow_add(struct kobject *kobj, struct sysfs_dirent
*shadow_parent)
 }

 /**
- * kobject_add - add an object to the hierarchy.
- * @kobj: object.
- */
-int kobject_add(struct kobject * kobj)
-{
- return kobject_shadow_add(kobj, NULL);
-}
-
-
-/**
  * kobject_register - initialize and add an object.
  * @kobj: object in question.
  */
@@ -338,7 +327,7 @@ int kobject_rename(struct kobject * kobj, const char *new_name)
  /* Note : if we want to send the new name alone, not the full path,
   * we could probably use kobject_name(kobj); */

- error = sysfs_rename_dir(kobj, kobj->parent->sd, new_name);
+ error = sysfs_rename_dir(kobj, new_name);

  /* This function is mostly/only used for network interface.
   * Some hotplug package track interfaces by their name and
@@ -355,27 +344,6 @@ out:
```

```
 }

 /**
- * kobject_rename - change the name of an object
- * @kobj: object in question.
- * @new_parent: object's new parent
- * @new_name: object's new name
- */
-
-int kobject_shadow_rename(struct kobject *kobj,
-     struct sysfs_dirent *new_parent, const char *new_name)
-{
- int error = 0;
-
- kobj = kobject_get(kobj);
- if (!kobj)
-  return -EINVAL;
- error = sysfs_rename_dir(kobj, new_parent, new_name);
- kobject_put(kobj);
-
- return error;
-}
-
-/**
  * kobject_move - move object to another parent
  * @kobj: object in question.
  * @new_parent: object's new parent (can be NULL)
--
1.5.1.1.181.g2de0
```

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,
  for (attr = grp->attrs; *attr && !error; attr++)
   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

Subject: [PATCH 3/4] sysfs: Implement sysfs_delete_link and sysfs_rename_link
Posted by ebiederm on Thu, 19 Jul 2007 04:46:39 GMT
View Forum Message <> Reply to Message

When removing a symlink sysfs_remove_link does not provide enough
information to figure out which shadow directory the symlink falls in.
So I need sysfs_delete_link which is passed the target of the symlink
to delete.

Further half the time when we are removing a symlink the code is
actually renaming the symlink but not doing so explicitly because we
don't have a symlink rename method. So I have added sysfs_rename_link
as well.

Both of these functions now have enough information to find a symlink
in a shadow directory. The only restriction is that they must be
called before the target kobject is renamed or deleted. If they are
called later I loose track of which tag the target kobject was marked
with and can no longer find the old symlink to remove it.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/sysfs/symlink.c | 31 +++++++++++++++++++++++++++++++
 include/linux/sysfs.h | 18 ++++++++++++++++++
 2 files changed, 49 insertions(+), 0 deletions(-)

diff --git a/fs/sysfs/symlink.c b/fs/sysfs/symlink.c
index e52d3ef..62b0aeb 100644
--- a/fs/sysfs/symlink.c
+++ b/fs/sysfs/symlink.c
@@ -126,6 +126,21 @@ addrm_finish:

 /**
+ * sysfs_delete_link - remove symlink in object's directory.
+ * @kobj: object we're acting for.
+ * @targ: object we're pointing to.
+ * @name: name of the symlink to remove.
+ *
+ * Unlike sysfs_remove_link sysfs_delete_link has enough information
+ * to successfully delete symlinks in shadow directories.
+ */
+void sysfs_delete_link(struct kobject *kobj, struct kobject *targ,
+   const char *name)
+{
+ sysfs_hash_and_remove(targ, kobj->sd, name);
+}
+
+/**

```
  * sysfs_remove_link - remove symlink in object's directory.
  * @kobj: object we're acting for.
  * @name: name of the symlink to remove.
@@ -136,6 +151,22 @@ void sysfs_remove_link(struct kobject * kobj, const char * name)
  sysfs_hash_and_remove(kobj, kobj->sd, name);
 }

+/**
+ * sysfs_rename_link - rename symlink in object's directory.
+ * @kobj: object we're acting for.
+ * @targ: object we're pointing to.
+ * @old: previous name of the symlink.
+ * @new: new name of the symlink.
+ *
+ * A helper function for the common rename symlink idiom.
+ */
+int sysfs_rename_link(struct kobject *kobj, struct kobject *targ,
+    const char *old, const char *new)
+{
+ sysfs_delete_link(kobj, targ, old);
+ return sysfs_create_link(kobj, targ, new);
+}
+
 static int sysfs_get_target_path(struct sysfs_dirent * parent_sd,
      struct sysfs_dirent * target_sd, char *path)
 {
diff --git a/include/linux/sysfs.h b/include/linux/sysfs.h
index 7cda047..e195cb5 100644
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -127,6 +127,13 @@ sysfs_create_link(struct kobject * kobj, struct kobject * target, const char
* n
 extern void
 sysfs_remove_link(struct kobject *, const char * name);

+extern int
+sysfs_rename_link(struct kobject *kobj, struct kobject *target,
+    const char *old_name, const char *new_name);
+
+extern void
+sysfs_delete_link(struct kobject *dir, struct kobject *targ, const char *name);
+
 int __must_check sysfs_create_bin_file(struct kobject *kobj,
      struct bin_attribute *attr);
 void sysfs_remove_bin_file(struct kobject *kobj, struct bin_attribute *attr);
@@ -202,6 +209,17 @@ static inline void sysfs_remove_link(struct kobject * k, const char *
name)
  ;
```

```
 }

+static inline int
+sysfs_rename_link(struct kobject * k, struct kobject *t,
+   const char *old_name, const char * new_name)
+{
+ return 0;
+}
+
+static inline void
+sysfs_delete_link(struct kobject *k, struct kobject *t, const char *name)
+{
+}

 static inline int sysfs_create_bin_file(struct kobject * k, struct bin_attribute * a)
 {
--
1.5.1.1.181.g2de0
```

Subject: [PATCH 4/4] driver core: Implement shadow directory support for device classes.

Posted by ebiederm on Thu, 19 Jul 2007 04:47:27 GMT

View Forum Message <> Reply to Message

This patch enables shadowing on every class directory if struct class
has shadow_ops.

In addition device_del and device_rename were modified to use
sysfs_delete_link and sysfs_rename_link respectively to ensure
when these operations happen on devices whos classes have
shadow operations that they work properly.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
---
 drivers/base/class.c   |   30 ++++++++++++++++++++++++++----
 drivers/base/core.c    |   45 ++++++++++++++++++++++++++++++---------------
 include/linux/device.h |    2 ++
 3 files changed, 52 insertions(+), 25 deletions(-)

diff --git a/drivers/base/class.c b/drivers/base/class.c
index 4d22226..c981f75 100644
--- a/drivers/base/class.c
```

```
+++ b/drivers/base/class.c
@@ -134,6 +134,17 @@ static void remove_class_attrs(struct class * cls)
  }
 }

+static int class_setup_shadowing(struct class *cls)
+{
+ const struct shadow_dir_operations *shadow_ops;
+
+ shadow_ops = cls->shadow_ops;
+ if (!shadow_ops)
+  return 0;
+
+ return sysfs_enable_shadowing(&cls->subsys.kobj, shadow_ops);
+}
+
 int class_register(struct class * cls)
 {
  int error;
@@ -152,11 +163,22 @@ int class_register(struct class * cls)
  subsys_set_kset(cls, class_subsys);

  error = subsystem_register(&cls->subsys);
- if (!error) {
-  error = add_class_attrs(class_get(cls));
-  class_put(cls);
- }
+ if (error)
+  goto out;
+
+ error = class_setup_shadowing(cls);
+ if (error)
+  goto out_unregister;
+
+ error = add_class_attrs(cls);
+ if (error)
+  goto out_unregister;
+
+out:
  return error;
+out_unregister:
+ subsystem_unregister(&cls->subsys);
+ goto out;
 }

 void class_unregister(struct class * cls)
diff --git a/drivers/base/core.c b/drivers/base/core.c
index 3599ab2..5b03842 100644
```

```
--- a/drivers/base/core.c
+++ b/drivers/base/core.c
@@ -639,8 +639,14 @@ static struct kobject * get_device_parent(struct device *dev,
    return kobj;

   /* or create a new class-directory at the parent device */
-  return kobject_kset_add_dir(&dev->class->class_dirs,
+  kobj = kobject_kset_add_dir(&dev->class->class_dirs,
          parent_kobj, dev->class->name);
+
+  /* If we created a new class-directory setup shadowing */
+  if (kobj && dev->class->shadow_ops)
+   sysfs_enable_shadowing(kobj, dev->class->shadow_ops);
+
+  return kobj;
  }

  if (parent)
@@ -937,8 +943,8 @@ void device_del(struct device * dev)
   /* If this is not a "fake" compatible device, remove the
    * symlink from the class to the device. */
   if (dev->kobj.parent != &dev->class->subsys.kobj)
-   sysfs_remove_link(&dev->class->subsys.kobj,
-       dev->bus_id);
+   sysfs_delete_link(&dev->class->subsys.kobj,
+       &dev->kobj, dev->bus_id);
   if (parent) {
 #ifdef CONFIG_SYSFS_DEPRECATED
    char *class_name = make_class_name(dev->class->name,
@@ -1236,6 +1242,13 @@ int device_rename(struct device *dev, char *new_name)
  strlcpy(old_device_name, dev->bus_id, BUS_ID_SIZE);
  strlcpy(dev->bus_id, new_name, BUS_ID_SIZE);

+ if (dev->class && (dev->kobj.parent != &dev->class->subsys.kobj)) {
+  error = sysfs_rename_link(&dev->class->subsys.kobj,
+   &dev->kobj, old_device_name, new_name);
+  if (error)
+   goto out;
+ }
+
  error = kobject_rename(&dev->kobj, new_name);
  if (error) {
   strlcpy(dev->bus_id, old_device_name, BUS_ID_SIZE);
@@ -1244,27 +1257,17 @@ int device_rename(struct device *dev, char *new_name)

 #ifdef CONFIG_SYSFS_DEPRECATED
  if (old_class_name) {
+  error = -ENOMEM;
```

```
   new_class_name = make_class_name(dev->class->name, &dev->kobj);
-  if (new_class_name) {
-   error = sysfs_create_link(&dev->parent->kobj,
-         &dev->kobj, new_class_name);
-   if (error)
-    goto out;
-   sysfs_remove_link(&dev->parent->kobj, old_class_name);
-  }
- }
-#endif
+  if (!new_class_name)
+   goto out;

- if (dev->class) {
-  sysfs_remove_link(&dev->class->subsys.kobj, old_device_name);
-  error = sysfs_create_link(&dev->class->subsys.kobj, &dev->kobj,
-        dev->bus_id);
-  if (error) {
-   /* Uh... how to unravel this if restoring can fail? */
-   dev_err(dev, "%s: sysfs_create_symlink failed (%d)\n",
-    __FUNCTION__, error);
-  }
+  error = sysfs_rename_link(&dev->parent->kobj, &dev->kobj,
+        old_class_name, new_class_name);
+  if (error)
+   goto out;
  }
+#endif
 out:
  put_device(dev);

diff --git a/include/linux/device.h b/include/linux/device.h
index d9f0a57..ed31e43 100644
--- a/include/linux/device.h
+++ b/include/linux/device.h
@@ -200,6 +200,8 @@ struct class {

  int (*suspend)(struct device *, pm_message_t state);
  int (*resume)(struct device *);
+
+ const struct shadow_dir_operations *shadow_ops;
 };

 extern int __must_check class_register(struct class *);
--
1.5.1.1.181.g2de0
```

Subject: patch
driver-core-implement-shadow-directory-support-for-device-classes.patch added to gregkh-2.6 tr
Posted by gregkh on Sat, 21 Jul 2007 06:36:02 GMT
View Forum Message <> Reply to Message

This is a note to let you know that I've just added the patch titled

    Subject: [PATCH 4/4] driver core: Implement shadow directory support for device classes.

to my gregkh-2.6 tree. Its filename is

    driver-core-implement-shadow-directory-support-for-device-classes.patch

This tree can be found at
   http://www.kernel.org/pub/linux/kernel/people/gregkh/gregkh-2.6/patches/

>From ebiederm@xmission.com Fri Jul 20 23:21:18 2007
From: ebiederm@xmission.com (Eric W. Biederman)
Date: Wed, 18 Jul 2007 22:47:27 -0600
Subject: [PATCH 4/4] driver core: Implement shadow directory support for device classes.
To: Greg KH <greg@kroah.com>
Cc: Greg KH <gregkh@suse.de>, Dave Hansen <hansendc@us.ibm.com>, Benjamin Thery <benjamin.thery@bull.net>, Linux Containers <containers@lists.osdl.org>, Tejun Heo <htejun@gmail.com>
Message-ID: <m1r6n52e5c.fsf_-_@ebiederm.dsl.xmission.com>

This patch enables shadowing on every class directory if struct class
has shadow_ops.

In addition device_del and device_rename were modified to use
sysfs_delete_link and sysfs_rename_link respectively to ensure
when these operations happen on devices whos classes have
shadow operations that they work properly.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
Cc: Tejun Heo <htejun@gmail.com>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

```
 drivers/base/class.c   |   30 ++++++++++++++++++++++++++----
 drivers/base/core.c    |   45 ++++++++++++++++++++++++++++++---------------
 include/linux/device.h |    2 ++
 3 files changed, 52 insertions(+), 25 deletions(-)

--- a/drivers/base/class.c
+++ b/drivers/base/class.c
@@ -134,6 +134,17 @@ static void remove_class_attrs(struct cl
  }
 }

+static int class_setup_shadowing(struct class *cls)
+{
+ const struct shadow_dir_operations *shadow_ops;
+
+ shadow_ops = cls->shadow_ops;
+ if (!shadow_ops)
+  return 0;
+
+ return sysfs_enable_shadowing(&cls->subsys.kobj, shadow_ops);
+}
+
 int class_register(struct class * cls)
 {
  int error;
@@ -152,11 +163,22 @@ int class_register(struct class * cls)
  subsys_set_kset(cls, class_subsys);

  error = subsystem_register(&cls->subsys);
- if (!error) {
-  error = add_class_attrs(class_get(cls));
-  class_put(cls);
- }
+ if (error)
+  goto out;
+
+ error = class_setup_shadowing(cls);
+ if (error)
+  goto out_unregister;
+
+ error = add_class_attrs(cls);
+ if (error)
+  goto out_unregister;
+
+out:
  return error;
+out_unregister:
+ subsystem_unregister(&cls->subsys);
```

```
+ goto out;
 }

 void class_unregister(struct class * cls)
--- a/drivers/base/core.c
+++ b/drivers/base/core.c
@@ -645,8 +645,14 @@ static struct kobject * get_device_paren
    return kobj;

   /* or create a new class-directory at the parent device */
-  return kobject_kset_add_dir(&dev->class->class_dirs,
+  kobj = kobject_kset_add_dir(&dev->class->class_dirs,
          parent_kobj, dev->class->name);
+
+  /* If we created a new class-directory setup shadowing */
+  if (kobj && dev->class->shadow_ops)
+   sysfs_enable_shadowing(kobj, dev->class->shadow_ops);
+
+  return kobj;
  }

  if (parent)
@@ -844,8 +850,8 @@ int device_add(struct device *dev)
   /* If this is not a "fake" compatible device, remove the
    * symlink from the class to the device. */
   if (dev->kobj.parent != &dev->class->subsys.kobj)
-   sysfs_remove_link(&dev->class->subsys.kobj,
-       dev->bus_id);
+   sysfs_delete_link(&dev->class->subsys.kobj,
+       &dev->kobj, dev->bus_id);
   if (parent && parent->bus) {
 #ifdef CONFIG_SYSFS_DEPRECATED
    char *class_name = make_class_name(dev->class->name,
@@ -1243,6 +1249,13 @@ int device_rename(struct device *dev, ch
  strlcpy(old_device_name, dev->bus_id, BUS_ID_SIZE);
  strlcpy(dev->bus_id, new_name, BUS_ID_SIZE);

+ if (dev->class && (dev->kobj.parent != &dev->class->subsys.kobj)) {
+  error = sysfs_rename_link(&dev->class->subsys.kobj,
+   &dev->kobj, old_device_name, new_name);
+  if (error)
+   goto out;
+ }
+
  error = kobject_rename(&dev->kobj, new_name);
  if (error) {
   strlcpy(dev->bus_id, old_device_name, BUS_ID_SIZE);
@@ -1251,27 +1264,17 @@ int device_rename(struct device *dev, ch
```

```
 #ifdef CONFIG_SYSFS_DEPRECATED
  if (old_class_name) {
+  error = -ENOMEM;
   new_class_name = make_class_name(dev->class->name, &dev->kobj);
-  if (new_class_name) {
-   error = sysfs_create_link(&dev->parent->kobj,
-         &dev->kobj, new_class_name);
-   if (error)
-    goto out;
-   sysfs_remove_link(&dev->parent->kobj, old_class_name);
-  }
- }
-#endif
+  if (!new_class_name)
+   goto out;

- if (dev->class) {
-  sysfs_remove_link(&dev->class->subsys.kobj, old_device_name);
-  error = sysfs_create_link(&dev->class->subsys.kobj, &dev->kobj,
-        dev->bus_id);
-  if (error) {
-   /* Uh... how to unravel this if restoring can fail? */
-   dev_err(dev, "%s: sysfs_create_symlink failed (%d)\n",
-    __FUNCTION__, error);
-  }
+  error = sysfs_rename_link(&dev->parent->kobj, &dev->kobj,
+        old_class_name, new_class_name);
+  if (error)
+   goto out;
  }
+#endif
 out:
  put_device(dev);

--- a/include/linux/device.h
+++ b/include/linux/device.h
@@ -200,6 +200,8 @@ struct class {

  int (*suspend)(struct device *, pm_message_t state);
  int (*resume)(struct device *);
+
+ const struct shadow_dir_operations *shadow_ops;
 };

 extern int __must_check class_register(struct class *);
```

Patches currently in gregkh-2.6 which might be from greg@kroah.com are

bad/pci-domain/pci-device-ensure-sysdata-initialised.patch
bad/pci-domain/pci-fix-the-x86-pci-domain-support-fix.patch
bad/relayfs/sysfs-update-relay-file-support-for-generic-relay-api.patch
bad/relayfs/relay-consolidate-relayfs-core-into-kernel-relay.c.patch
bad/relayfs/relay-relay-header-cleanup.patch
bad/relayfs/relayfs-remove-relayfs-in-favour-of-config_relay.patch
bad/relayfs/sysfs-add-__attr_relay-helper-for-relay-attributes.patch
bad/relayfs/sysfs-relay-channel-buffers-as-sysfs-attributes.patch
bad/usbip/usb-usbip-more-dead-code-fix.patch
bad/usbip/usb-usbip-build-fix.patch
bad/usbip/usb-usbip-warning-fixes.patch
bad/ndevfs.patch
bad/battery-class-driver.patch
bad/driver-model-convert-driver-model-to-mutexes.patch
bad/gpl_future-test.patch
bad/gregkh-debugfs_example.patch
bad/speakup-kconfig-fix.patch
bad/speakup-build-fix.patch
bad/pci-use-new-multi-phase-suspend-infrastructure.patch
bad/shot-accross-the-bow.patch
bad/no-more-non-gpl-modules.patch
bad/spi-device.patch
bad/ata_piix-multithread.patch
bad/uio-irq.patch
bad/pci-two-drivers-on-one-pci-device.patch
bad/pci-dynamic-id-cleanup.patch
bad/input-device.patch
bad/usb-stimulus.patch
driver/nozomi.patch
driver/kobject-put-kobject_actions-in-kobject.h.patch
driver/sysfs-implement-sysfs-manged-shadow-directory-support.patch
driver/sysfs-implement-sysfs_delete_link-and-sysfs_rename_link.patch
driver/sysfs-remove-first-pass-at-shadow-directory-support.patch
driver/driver-core-implement-shadow-directory-support-for-device-classes.patch
gregkh/gkh-version.patch
gregkh/sysfs-test.patch
gregkh/sysrq-u-laptop.patch
pci/pci_bridge-device.patch
pci/pci-piggy-bus.patch
pci/pci-move-prototypes-for-pci_bus_find_capability-to-include-linux-pci.h.patch
pci/pci-document-pci_iomap.patch
usb/usb-gotemp.patch
usb/kobject-put-kobject_actions-in-kobject.h.patch
usb/usb-add-the-concept-of-default-authorization-to-usb-hosts.patch
usb/usb-cleanup-usb_register_bus-and-hook-up-sysfs-group.patch
usb/usb-initialize-authorization-and-wusb-bits-in-usb-devices.patch

usb/usb-introduce-usb_device-authorization-bits.patch
usb/usb-usb_set_configuration-obeys-authorization.patch
usb/usb-usb.h-kernel-doc-additions.patch
HOWTO

_____

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

---

Subject: patch sysfs-implement-sysfs-manged-shadow-directory-support.patch added to gregkh-2.6 tree
Posted by gregkh on Sat, 21 Jul 2007 06:36:16 GMT
View Forum Message <> Reply to Message

This is a note to let you know that I've just added the patch titled

    Subject: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.

to my gregkh-2.6 tree. Its filename is

    sysfs-implement-sysfs-manged-shadow-directory-support.patch

This tree can be found at
   http://www.kernel.org/pub/linux/kernel/people/gregkh/gregkh-2.6/patches/

>From ebiederm@xmission.com Fri Jul 20 23:20:42 2007
From: ebiederm@xmission.com (Eric W. Biederman)
Date: Wed, 18 Jul 2007 22:45:13 -0600
Subject: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.
To: Greg KH <greg@kroah.com>
Cc: Greg KH <gregkh@suse.de>, Dave Hansen <hansendc@us.ibm.com>, Benjamin Thery <benjamin.thery@bull.net>, Linux Containers <containers@lists.osdl.org>, Tejun Heo <htejun@gmail.com>
Message-ID: <m1zm1t2e92.fsf_-_@ebiederm.dsl.xmission.com>

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>
Cc: Tejun Heo <htejun@gmail.com>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

---
 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(-)

--- a/fs/sysfs/bin.c
+++ b/fs/sysfs/bin.c
@@ -248,7 +248,7 @@ int sysfs_create_bin_file(struct kobject

 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);
--- 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_d
  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 s
  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 sys
  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_addr
   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 *ko
  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 *ko

  /* 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(stru
  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_di
  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 sy
   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 * k

 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 * f
    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_o
  .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;
+}
+
--- 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

  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);
  }
 }
--- 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_dir
  for (attr = grp->attrs; *attr && !error; attr++)
   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 *
  } 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 *
  } else
   sd = sysfs_get(dir_sd);

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

--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -187,17 +187,16 @@ void sysfs_instantiate(struct dentry *de
  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_d
    break;
   }
  }
-
+addrm_finish:
  if (sysfs_addrm_finish(&acxt))
   return 0;
  return -ENOENT;
--- a/fs/sysfs/symlink.c
+++ b/fs/sysfs/symlink.c
@@ -15,8 +15,11 @@ static int object_depth(struct sysfs_dir
 {
  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 sys
 {
  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 * k
  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 * k

 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,
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -58,6 +58,12 @@ extern struct kmem_cache *sysfs_dir_cach
 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_di

 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,
--- 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

 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 k
 {
 }

+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;
```

Patches currently in gregkh-2.6 which might be from greg@kroah.com are

bad/pci-domain/pci-device-ensure-sysdata-initialised.patch
bad/pci-domain/pci-fix-the-x86-pci-domain-support-fix.patch
bad/relayfs/sysfs-update-relay-file-support-for-generic-relay-api.patch
bad/relayfs/relay-consolidate-relayfs-core-into-kernel-relay.c.patch
bad/relayfs/relay-relay-header-cleanup.patch
bad/relayfs/relayfs-remove-relayfs-in-favour-of-config_relay.patch
bad/relayfs/sysfs-add-__attr_relay-helper-for-relay-attributes.patch
bad/relayfs/sysfs-relay-channel-buffers-as-sysfs-attributes.patch
bad/usbip/usb-usbip-more-dead-code-fix.patch
bad/usbip/usb-usbip-build-fix.patch
bad/usbip/usb-usbip-warning-fixes.patch
bad/ndevfs.patch
bad/battery-class-driver.patch
bad/driver-model-convert-driver-model-to-mutexes.patch
bad/gpl_future-test.patch
bad/gregkh-debugfs_example.patch
bad/speakup-kconfig-fix.patch
bad/speakup-build-fix.patch
bad/pci-use-new-multi-phase-suspend-infrastructure.patch
bad/shot-accross-the-bow.patch
bad/no-more-non-gpl-modules.patch
bad/spi-device.patch
bad/ata_piix-multithread.patch
bad/uio-irq.patch
bad/pci-two-drivers-on-one-pci-device.patch

bad/pci-dynamic-id-cleanup.patch
bad/input-device.patch
bad/usb-stimulus.patch
driver/nozomi.patch
driver/kobject-put-kobject_actions-in-kobject.h.patch
driver/sysfs-implement-sysfs-manged-shadow-directory-support.patch
driver/sysfs-implement-sysfs_delete_link-and-sysfs_rename_link.patch
driver/sysfs-remove-first-pass-at-shadow-directory-support.patch
driver/driver-core-implement-shadow-directory-support-for-device-classes.patch
gregkh/gkh-version.patch
gregkh/sysfs-test.patch
gregkh/sysrq-u-laptop.patch
pci/pci_bridge-device.patch
pci/pci-piggy-bus.patch
pci/pci-move-prototypes-for-pci_bus_find_capability-to-include-linux-pci.h.patch
pci/pci-document-pci_iomap.patch
usb/usb-gotemp.patch
usb/kobject-put-kobject_actions-in-kobject.h.patch
usb/usb-add-the-concept-of-default-authorization-to-usb-hosts.patch
usb/usb-cleanup-usb_register_bus-and-hook-up-sysfs-group.patch
usb/usb-initialize-authorization-and-wusb-bits-in-usb-devices.patch
usb/usb-introduce-usb_device-authorization-bits.patch
usb/usb-usb_set_configuration-obeys-authorization.patch
usb/usb-usb.h-kernel-doc-additions.patch
HOWTO

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

---

Subject: patch sysfs-implement-sysfs_delete_link-and-sysfs_rename_link.patch added to gregkh-2.6 tree
Posted by gregkh on Sat, 21 Jul 2007 06:36:23 GMT
View Forum Message <> Reply to Message

This is a note to let you know that I've just added the patch titled

    Subject: [PATCH 3/4] sysfs: Implement sysfs_delete_link and sysfs_rename_link

to my gregkh-2.6 tree. Its filename is

    sysfs-implement-sysfs_delete_link-and-sysfs_rename_link.patch

This tree can be found at
   http://www.kernel.org/pub/linux/kernel/people/gregkh/gregkh-2.6/patches/

>From ebiederm@xmission.com Fri Jul 20 23:21:06 2007
From: ebiederm@xmission.com (Eric W. Biederman)
Date: Wed, 18 Jul 2007 22:46:39 -0600
Subject: [PATCH 3/4] sysfs: Implement sysfs_delete_link and sysfs_rename_link
To: Greg KH <greg@kroah.com>
Cc: Greg KH <gregkh@suse.de>, Dave Hansen <hansendc@us.ibm.com>, Benjamin Thery <benjamin.thery@bull.net>, Linux Containers <containers@lists.osdl.org>, Tejun Heo <htejun@gmail.com>
Message-ID: <m1vech2e6o.fsf_-_@ebiederm.dsl.xmission.com>

When removing a symlink sysfs_remove_link does not provide enough
information to figure out which shadow directory the symlink falls in.
So I need sysfs_delete_link which is passed the target of the symlink
to delete.

Further half the time when we are removing a symlink the code is
actually renaming the symlink but not doing so explicitly because we
don't have a symlink rename method. So I have added sysfs_rename_link
as well.

Both of these functions now have enough information to find a symlink
in a shadow directory. The only restriction is that they must be
called before the target kobject is renamed or deleted. If they are
called later I loose track of which tag the target kobject was marked
with and can no longer find the old symlink to remove it.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
Cc: Tejun Heo <htejun@gmail.com>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

---
 fs/sysfs/symlink.c | 31 ++++++++++++++++++++++++++++++++++
 include/linux/sysfs.h | 18 ++++++++++++++++++
 2 files changed, 49 insertions(+)

--- a/fs/sysfs/symlink.c
+++ b/fs/sysfs/symlink.c
@@ -126,6 +126,21 @@ addrm_finish:

 /**
+ * sysfs_delete_link - remove symlink in object's directory.
+ * @kobj: object we're acting for.
+ * @targ: object we're pointing to.
+ * @name: name of the symlink to remove.
+ *

```
+ * Unlike sysfs_remove_link sysfs_delete_link has enough information
+ * to successfully delete symlinks in shadow directories.
+ */
+void sysfs_delete_link(struct kobject *kobj, struct kobject *targ,
+   const char *name)
+{
+ sysfs_hash_and_remove(targ, kobj->sd, name);
+}
+
+/**
  * sysfs_remove_link - remove symlink in object's directory.
  * @kobj: object we're acting for.
  * @name: name of the symlink to remove.
@@ -136,6 +151,22 @@ void sysfs_remove_link(struct kobject *
  sysfs_hash_and_remove(kobj, kobj->sd, name);
 }

+/**
+ * sysfs_rename_link - rename symlink in object's directory.
+ * @kobj: object we're acting for.
+ * @targ: object we're pointing to.
+ * @old: previous name of the symlink.
+ * @new: new name of the symlink.
+ *
+ * A helper function for the common rename symlink idiom.
+ */
+int sysfs_rename_link(struct kobject *kobj, struct kobject *targ,
+   const char *old, const char *new)
+{
+ sysfs_delete_link(kobj, targ, old);
+ return sysfs_create_link(kobj, targ, new);
+}
+
 static int sysfs_get_target_path(struct sysfs_dirent * parent_sd,
      struct sysfs_dirent * target_sd, char *path)
 {
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -127,6 +127,13 @@ sysfs_create_link(struct kobject * kobj,
 extern void
 sysfs_remove_link(struct kobject *, const char * name);

+extern int
+sysfs_rename_link(struct kobject *kobj, struct kobject *target,
+   const char *old_name, const char *new_name);
+
+extern void
+sysfs_delete_link(struct kobject *dir, struct kobject *targ, const char *name);
```

```
+
 int __must_check sysfs_create_bin_file(struct kobject *kobj,
      struct bin_attribute *attr);
 void sysfs_remove_bin_file(struct kobject *kobj, struct bin_attribute *attr);
@@ -202,6 +209,17 @@ static inline void sysfs_remove_link(str
  ;
 }

+static inline int
+sysfs_rename_link(struct kobject * k, struct kobject *t,
+    const char *old_name, const char * new_name)
+{
+ return 0;
+}
+
+static inline void
+sysfs_delete_link(struct kobject *k, struct kobject *t, const char *name)
+{
+}

 static inline int sysfs_create_bin_file(struct kobject * k, struct bin_attribute * a)
 {
```

Patches currently in gregkh-2.6 which might be from greg@kroah.com are

bad/pci-domain/pci-device-ensure-sysdata-initialised.patch
bad/pci-domain/pci-fix-the-x86-pci-domain-support-fix.patch
bad/relayfs/sysfs-update-relay-file-support-for-generic-relay-api.patch
bad/relayfs/relay-consolidate-relayfs-core-into-kernel-relay.c.patch
bad/relayfs/relay-relay-header-cleanup.patch
bad/relayfs/relayfs-remove-relayfs-in-favour-of-config_relay.patch
bad/relayfs/sysfs-add-__attr_relay-helper-for-relay-attributes.patch
bad/relayfs/sysfs-relay-channel-buffers-as-sysfs-attributes.patch
bad/usbip/usb-usbip-more-dead-code-fix.patch
bad/usbip/usb-usbip-build-fix.patch
bad/usbip/usb-usbip-warning-fixes.patch
bad/ndevfs.patch
bad/battery-class-driver.patch
bad/driver-model-convert-driver-model-to-mutexes.patch
bad/gpl_future-test.patch
bad/gregkh-debugfs_example.patch
bad/speakup-kconfig-fix.patch
bad/speakup-build-fix.patch
bad/pci-use-new-multi-phase-suspend-infrastructure.patch
bad/shot-accross-the-bow.patch
bad/no-more-non-gpl-modules.patch
bad/spi-device.patch

bad/ata_piix-multithread.patch
bad/uio-irq.patch
bad/pci-two-drivers-on-one-pci-device.patch
bad/pci-dynamic-id-cleanup.patch
bad/input-device.patch
bad/usb-stimulus.patch
driver/nozomi.patch
driver/kobject-put-kobject_actions-in-kobject.h.patch
driver/sysfs-implement-sysfs-manged-shadow-directory-support.patch
driver/sysfs-implement-sysfs_delete_link-and-sysfs_rename_link.patch
driver/sysfs-remove-first-pass-at-shadow-directory-support.patch
driver/driver-core-implement-shadow-directory-support-for-device-classes.patch
gregkh/gkh-version.patch
gregkh/sysfs-test.patch
gregkh/sysrq-u-laptop.patch
pci/pci_bridge-device.patch
pci/pci-piggy-bus.patch
pci/pci-move-prototypes-for-pci_bus_find_capability-to-include-linux-pci.h.patch
pci/pci-document-pci_iomap.patch
usb/usb-gotemp.patch
usb/kobject-put-kobject_actions-in-kobject.h.patch
usb/usb-add-the-concept-of-default-authorization-to-usb-hosts.patch
usb/usb-cleanup-usb_register_bus-and-hook-up-sysfs-group.patch
usb/usb-initialize-authorization-and-wusb-bits-in-usb-devices.patch
usb/usb-introduce-usb_device-authorization-bits.patch
usb/usb-usb_set_configuration-obeys-authorization.patch
usb/usb-usb.h-kernel-doc-additions.patch
HOWTO

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

---

Subject: patch sysfs-remove-first-pass-at-shadow-directory-support.patch added to gregkh-2.6 tree
Posted by gregkh on Sat, 21 Jul 2007 06:36:28 GMT
View Forum Message <> Reply to Message

This is a note to let you know that I've just added the patch titled

    Subject: [PATCH 1/4] sysfs: Remove first pass at shadow directory support

to my gregkh-2.6 tree. Its filename is

    sysfs-remove-first-pass-at-shadow-directory-support.patch

This tree can be found at

http://www.kernel.org/pub/linux/kernel/people/gregkh/gregkh-2.6/patches/

>From ebiederm@xmission.com Fri Jul 20 23:20:07 2007
From: ebiederm@xmission.com (Eric W. Biederman)
Date: Wed, 18 Jul 2007 22:43:46 -0600
Subject: [PATCH 1/4] sysfs: Remove first pass at shadow directory support
To: Greg KH <greg@kroah.com>
Cc: Greg KH <gregkh@suse.de>, Dave Hansen <hansendc@us.ibm.com>, Benjamin Thery <benjamin.thery@bull.net>, Linux Containers <containers@lists.osdl.org>, Tejun Heo <htejun@gmail.com>
Message-ID: <m14pk13svx.fsf@ebiederm.dsl.xmission.com>

While shadow directories appear to be a good idea, the current scheme
of controlling their creation and destruction outside of sysfs appears
to be a locking and maintenance nightmare in the face of sysfs
directories dynamically coming and going. Which can now occur for
directories containing network devices when CONFIG_SYSFS_DEPRECATED is
not set.

This patch removes everything from the initial shadow directory support
that allowed the shadow directory creation to be controlled at a higher
level. So except for a few bits of sysfs_rename_dir everything from
commit b592fcfe7f06c15ec11774b5be7ce0de3aa86e73 is now gone.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

---
 fs/sysfs/dir.c | 167 ++++++--
 fs/sysfs/group.c | 1
 fs/sysfs/inode.c | 10 --
 fs/sysfs/mount.c | 2
 fs/sysfs/sysfs.h | 6 -
 include/linux/kobject.h | 5 -
 include/linux/sysfs.h | 27 +------
 lib/kobject.c | 44 +----------
 8 files changed, 33 insertions(+), 229 deletions(-)

--- a/fs/sysfs/dir.c
+++ b/fs/sysfs/dir.c
@@ -569,9 +569,6 @@ static void sysfs_drop_dentry(struct sys
  spin_unlock(&dcache_lock);
  spin_unlock(&sysfs_assoc_lock);

- /* dentries for shadowed inodes are pinned, unpin */

```
- if (dentry && sysfs_is_shadowed_inode(dentry->d_inode))
-  dput(dentry);
  dput(dentry);

  /* adjust nlink and update timestamp */
@@ -723,19 +720,15 @@ int sysfs_create_subdir(struct kobject *
 /**
  * sysfs_create_dir - create a directory for an object.
  * @kobj:  object we're creating directory for.
- * @shadow_parent: parent object.
  */
-int sysfs_create_dir(struct kobject *kobj,
-       struct sysfs_dirent *shadow_parent_sd)
+int sysfs_create_dir(struct kobject * kobj)
 {
  struct sysfs_dirent *parent_sd, *sd;
  int error = 0;

  BUG_ON(!kobj);

- if (shadow_parent_sd)
-  parent_sd = shadow_parent_sd;
- else if (kobj->parent)
+ if (kobj->parent)
   parent_sd = kobj->parent->sd;
  else if (sysfs_mount && sysfs_mount->mnt_sb)
   parent_sd = sysfs_mount->mnt_sb->s_root->d_fsdata;
@@ -887,45 +880,44 @@ void sysfs_remove_dir(struct kobject * k
  __sysfs_remove_dir(sd);
 }

-int sysfs_rename_dir(struct kobject *kobj, struct sysfs_dirent *new_parent_sd,
-       const char *new_name)
+int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
 {
- struct sysfs_dirent *sd = kobj->sd;
- struct dentry *new_parent = NULL;
+ 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;
  int error;

+ if (!kobj->parent)
+  return -EINVAL;
+
  /* get dentries */
```

```
+ sd = kobj->sd;
  old_dentry = sysfs_get_dentry(sd);
  if (IS_ERR(old_dentry)) {
   error = PTR_ERR(old_dentry);
   goto out_dput;
  }

- new_parent = sysfs_get_dentry(new_parent_sd);
- if (IS_ERR(new_parent)) {
-  error = PTR_ERR(new_parent);
+ parent_sd = kobj->parent->sd;
+ parent = sysfs_get_dentry(parent_sd);
+ if (IS_ERR(parent)) {
+  error = PTR_ERR(parent);
   goto out_dput;
  }

- /* lock new_parent and get dentry for new name */
- mutex_lock(&new_parent->d_inode->i_mutex);
+ /* lock parent and get dentry for new name */
+ mutex_lock(&parent->d_inode->i_mutex);

- new_dentry = lookup_one_len(new_name, new_parent, strlen(new_name));
+ new_dentry = lookup_one_len(new_name, parent, strlen(new_name));
  if (IS_ERR(new_dentry)) {
   error = PTR_ERR(new_dentry);
   goto out_unlock;
  }

- /* By allowing two different directories with the same
-  * d_parent we allow this routine to move between different
-  * shadows of the same directory
-  */
  error = -EINVAL;
- if (old_dentry->d_parent->d_inode != new_parent->d_inode ||
-     new_dentry->d_parent->d_inode != new_parent->d_inode ||
-     old_dentry == new_dentry)
+ if (old_dentry == new_dentry)
   goto out_unlock;

  error = -EEXIST;
@@ -952,9 +944,9 @@ int sysfs_rename_dir(struct kobject *kob
  mutex_lock(&sysfs_mutex);

  sysfs_unlink_sibling(sd);
- sysfs_get(new_parent_sd);
+ sysfs_get(parent_sd);
  sysfs_put(sd->s_parent);
```

```
- sd->s_parent = new_parent_sd;
+ sd->s_parent = parent_sd;
  sysfs_link_sibling(sd);

  mutex_unlock(&sysfs_mutex);
@@ -965,10 +957,10 @@ int sysfs_rename_dir(struct kobject *kob
  out_drop:
  d_drop(new_dentry);
  out_unlock:
- mutex_unlock(&new_parent->d_inode->i_mutex);
+ mutex_unlock(&parent->d_inode->i_mutex);
  out_dput:
  kfree(dup_name);
- dput(new_parent);
+ dput(parent);
  dput(old_dentry);
  dput(new_dentry);
  return error;
@@ -1189,121 +1181,6 @@ static loff_t sysfs_dir_lseek(struct fil
  return offset;
 }

-
-/**
- * sysfs_make_shadowed_dir - Setup so a directory can be shadowed
- * @kobj: object we're creating shadow of.
- */
-
-int sysfs_make_shadowed_dir(struct kobject *kobj,
- void * (*follow_link)(struct dentry *, struct nameidata *))
-{
- struct dentry *dentry;
- struct inode *inode;
- struct inode_operations *i_op;
-
- /* get dentry for @kobj->sd, dentry of a shadowed dir is pinned */
- dentry = sysfs_get_dentry(kobj->sd);
- if (IS_ERR(dentry))
-  return PTR_ERR(dentry);
-
- inode = dentry->d_inode;
- if (inode->i_op != &sysfs_dir_inode_operations) {
-  dput(dentry);
-  return -EINVAL;
- }
-
- i_op = kmalloc(sizeof(*i_op), GFP_KERNEL);
- if (!i_op)
```

```
-  return -ENOMEM;
-
- memcpy(i_op, &sysfs_dir_inode_operations, sizeof(*i_op));
- i_op->follow_link = follow_link;
-
- /* Locking of inode->i_op?
-  * Since setting i_op is a single word write and they
-  * are atomic we should be ok here.
-  */
- inode->i_op = i_op;
- return 0;
-}
-
-/**
- * sysfs_create_shadow_dir - create a shadow directory for an object.
- * @kobj: object we're creating directory for.
- *
- * sysfs_make_shadowed_dir must already have been called on this
- * directory.
- */
-
-struct sysfs_dirent *sysfs_create_shadow_dir(struct kobject *kobj)
-{
- struct sysfs_dirent *parent_sd = kobj->sd->s_parent;
- struct dentry *dir, *parent, *shadow;
- struct inode *inode;
- struct sysfs_dirent *sd;
- struct sysfs_addrm_cxt acxt;
-
- dir = sysfs_get_dentry(kobj->sd);
- if (IS_ERR(dir)) {
-  sd = (void *)dir;
-  goto out;
- }
- parent = dir->d_parent;
-
- inode = dir->d_inode;
- sd = ERR_PTR(-EINVAL);
- if (!sysfs_is_shadowed_inode(inode))
-  goto out_dput;
-
- shadow = d_alloc(parent, &dir->d_name);
- if (!shadow)
-  goto nomem;
-
- sd = sysfs_new_dirent("_SHADOW_", inode->i_mode, SYSFS_DIR);
- if (!sd)
-  goto nomem;
```

```
- sd->s_elem.dir.kobj = kobj;
-
- sysfs_addrm_start(&acxt, parent_sd);
-
- /* add but don't link into children list */
- sysfs_add_one(&acxt, sd);
-
- /* attach and instantiate dentry */
- sysfs_attach_dentry(sd, shadow);
- d_instantiate(shadow, igrab(inode));
- inc_nlink(inode); /* tj: synchronization? */
-
- sysfs_addrm_finish(&acxt);
-
- dget(shadow);  /* Extra count - pin the dentry in core */
-
- goto out_dput;
-
- nomem:
- dput(shadow);
- sd = ERR_PTR(-ENOMEM);
- out_dput:
- dput(dir);
- out:
- return sd;
-}
-
-/**
- * sysfs_remove_shadow_dir - remove an object's directory.
- * @shadow_sd: sysfs_dirent of shadow directory
- *
- * The only thing special about this is that we remove any files in
- * the directory before we remove the directory, and we've inlined
- * what used to be sysfs_rmdir() below, instead of calling separately.
- */
-
-void sysfs_remove_shadow_dir(struct sysfs_dirent *shadow_sd)
-{
- __sysfs_remove_dir(shadow_sd);
-}
-
 const struct file_operations sysfs_dir_operations = {
  .open  = sysfs_dir_open,
  .release = sysfs_dir_close,
--- a/fs/sysfs/group.c
+++ b/fs/sysfs/group.c
@@ -13,7 +13,6 @@
 #include <linux/dcache.h>
```

```
 #include <linux/namei.h>
 #include <linux/err.h>
-#include <linux/fs.h>
 #include <asm/semaphore.h>
 #include "sysfs.h"

--- a/fs/sysfs/inode.c
+++ b/fs/sysfs/inode.c
@@ -34,16 +34,6 @@ static const struct inode_operations sys
  .setattr = sysfs_setattr,
 };

-void sysfs_delete_inode(struct inode *inode)
-{
- /* Free the shadowed directory inode operations */
- if (sysfs_is_shadowed_inode(inode)) {
-  kfree(inode->i_op);
-  inode->i_op = NULL;
- }
- return generic_delete_inode(inode);
-}
-
 int sysfs_setattr(struct dentry * dentry, struct iattr * iattr)
 {
  struct inode * inode = dentry->d_inode;
--- a/fs/sysfs/mount.c
+++ b/fs/sysfs/mount.c
@@ -21,7 +21,7 @@ struct kmem_cache *sysfs_dir_cachep;

 static const struct super_operations sysfs_ops = {
  .statfs  = simple_statfs,
- .drop_inode = sysfs_delete_inode,
+ .drop_inode = generic_delete_inode,
 };

 struct sysfs_dirent sysfs_root = {
--- a/fs/sysfs/sysfs.h
+++ b/fs/sysfs/sysfs.h
@@ -70,7 +70,6 @@ extern void sysfs_remove_one(struct sysf
        struct sysfs_dirent *sd);
 extern int sysfs_addrm_finish(struct sysfs_addrm_cxt *acxt);

-extern void sysfs_delete_inode(struct inode *inode);
 extern struct inode * sysfs_get_inode(struct sysfs_dirent *sd);
 extern void sysfs_instantiate(struct dentry *dentry, struct inode *inode);

@@ -121,8 +120,3 @@ static inline void sysfs_put(struct sysf
  if (sd && atomic_dec_and_test(&sd->s_count))
```

```
  release_sysfs_dirent(sd);
 }
-
-static inline int sysfs_is_shadowed_inode(struct inode *inode)
-{
- return S_ISDIR(inode->i_mode) && inode->i_op->follow_link;
-}
--- a/include/linux/kobject.h
+++ b/include/linux/kobject.h
@@ -83,14 +83,9 @@ extern void kobject_init(struct kobject
 extern void kobject_cleanup(struct kobject *);

 extern int __must_check kobject_add(struct kobject *);
-extern int __must_check kobject_shadow_add(struct kobject *kobj,
-       struct sysfs_dirent *shadow_parent);
 extern void kobject_del(struct kobject *);

 extern int __must_check kobject_rename(struct kobject *, const char *new_name);
-extern int __must_check kobject_shadow_rename(struct kobject *kobj,
-          struct sysfs_dirent *new_parent,
-          const char *new_name);
 extern int __must_check kobject_move(struct kobject *, struct kobject *);

 extern int __must_check kobject_register(struct kobject *);
--- a/include/linux/sysfs.h
+++ b/include/linux/sysfs.h
@@ -17,9 +17,6 @@

 struct kobject;
 struct module;
-struct nameidata;
-struct dentry;
-struct sysfs_dirent;

 /* FIXME
  * The *owner field is no longer used, but leave around
@@ -94,14 +91,13 @@ extern int sysfs_schedule_callback(struc
   void (*func)(void *), void *data, struct module *owner);

 extern int __must_check
-sysfs_create_dir(struct kobject *kobj, struct sysfs_dirent *shadow_parent_sd);
+sysfs_create_dir(struct kobject *);

 extern void
 sysfs_remove_dir(struct kobject *);

 extern int __must_check
-sysfs_rename_dir(struct kobject *kobj, struct sysfs_dirent *new_parent_sd,
```

```
-   const char *new_name);
+sysfs_rename_dir(struct kobject *kobj, const char *new_name);

 extern int __must_check
 sysfs_move_dir(struct kobject *, struct kobject *);
@@ -138,12 +134,6 @@ void sysfs_remove_file_from_group(struct

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

-
-extern int sysfs_make_shadowed_dir(struct kobject *kobj,
- void * (*follow_link)(struct dentry *, struct nameidata *));
-extern struct sysfs_dirent *sysfs_create_shadow_dir(struct kobject *kobj);
-extern void sysfs_remove_shadow_dir(struct sysfs_dirent *shadow_sd);
-
 extern int __must_check sysfs_init(void);

 #else /* CONFIG_SYSFS */
@@ -154,8 +144,7 @@ static inline int sysfs_schedule_callbac
  return -ENOSYS;
 }

-static inline int sysfs_create_dir(struct kobject *kobj,
-       struct sysfs_dirent *shadow_parent_sd)
+static inline int sysfs_create_dir(struct kobject * kobj)
 {
  return 0;
 }
@@ -165,9 +154,7 @@ static inline void sysfs_remove_dir(stru
  ;
 }

-static inline int sysfs_rename_dir(struct kobject *kobj,
-       struct sysfs_dirent *new_parent_sd,
-       const char *new_name)
+static inline int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
 {
  return 0;
 }
@@ -242,12 +229,6 @@ static inline void sysfs_notify(struct k
 {
 }

-static inline int sysfs_make_shadowed_dir(struct kobject *kobj,
- void * (*follow_link)(struct dentry *, struct nameidata *))
-{
- return 0;
-}
```

```
-
 static inline int __must_check sysfs_init(void)
 {
  return 0;
--- a/lib/kobject.c
+++ b/lib/kobject.c
@@ -46,11 +46,11 @@ static int populate_dir(struct kobject *
  return error;
 }

-static int create_dir(struct kobject *kobj, struct sysfs_dirent *shadow_parent)
+static int create_dir(struct kobject * kobj)
 {
  int error = 0;
  if (kobject_name(kobj)) {
-  error = sysfs_create_dir(kobj, shadow_parent);
+  error = sysfs_create_dir(kobj);
   if (!error) {
    if ((error = populate_dir(kobj)))
     sysfs_remove_dir(kobj);
@@ -201,12 +201,11 @@ static void unlink(struct kobject * kobj
 }

 /**
- * kobject_shadow_add - add an object to the hierarchy.
+ * kobject_add - add an object to the hierarchy.
  * @kobj: object.
- * @shadow_parent: sysfs directory to add to.
  */

-int kobject_shadow_add(struct kobject *kobj, struct sysfs_dirent *shadow_parent)
+int kobject_add(struct kobject * kobj)
 {
  int error = 0;
  struct kobject * parent;
@@ -238,7 +237,7 @@ int kobject_shadow_add(struct kobject *k
   kobj->parent = parent;
  }

- error = create_dir(kobj, shadow_parent);
+ error = create_dir(kobj);
  if (error) {
   /* unlink does the kobject_put() for us */
   unlink(kobj);
@@ -260,16 +259,6 @@ int kobject_shadow_add(struct kobject *k
 }

 /**
```

```
- * kobject_add - add an object to the hierarchy.
- * @kobj: object.
- */
-int kobject_add(struct kobject * kobj)
-{
- return kobject_shadow_add(kobj, NULL);
-}
-
-
-/**
  * kobject_register - initialize and add an object.
  * @kobj: object in question.
  */
@@ -382,7 +371,7 @@ int kobject_rename(struct kobject * kobj
  /* Note : if we want to send the new name alone, not the full path,
   * we could probably use kobject_name(kobj); */

- error = sysfs_rename_dir(kobj, kobj->parent->sd, new_name);
+ error = sysfs_rename_dir(kobj, new_name);

  /* This function is mostly/only used for network interface.
   * Some hotplug package track interfaces by their name and
@@ -399,27 +388,6 @@ out:
 }

 /**
- * kobject_rename - change the name of an object
- * @kobj: object in question.
- * @new_parent: object's new parent
- * @new_name: object's new name
- */
-
-int kobject_shadow_rename(struct kobject *kobj,
-     struct sysfs_dirent *new_parent, const char *new_name)
-{
- int error = 0;
-
- kobj = kobject_get(kobj);
- if (!kobj)
-  return -EINVAL;
- error = sysfs_rename_dir(kobj, new_parent, new_name);
- kobject_put(kobj);
-
- return error;
-}
-
-/**
  * kobject_move - move object to another parent
```

```
 * @kobj: object in question.
 * @new_parent: object's new parent (can be NULL)
```

Patches currently in gregkh-2.6 which might be from greg@kroah.com are

bad/pci-domain/pci-device-ensure-sysdata-initialised.patch
bad/pci-domain/pci-fix-the-x86-pci-domain-support-fix.patch
bad/relayfs/sysfs-update-relay-file-support-for-generic-relay-api.patch
bad/relayfs/relay-consolidate-relayfs-core-into-kernel-relay.c.patch
bad/relayfs/relay-relay-header-cleanup.patch
bad/relayfs/relayfs-remove-relayfs-in-favour-of-config_relay.patch
bad/relayfs/sysfs-add-__attr_relay-helper-for-relay-attributes.patch
bad/relayfs/sysfs-relay-channel-buffers-as-sysfs-attributes.patch
bad/usbip/usb-usbip-more-dead-code-fix.patch
bad/usbip/usb-usbip-build-fix.patch
bad/usbip/usb-usbip-warning-fixes.patch
bad/ndevfs.patch
bad/battery-class-driver.patch
bad/driver-model-convert-driver-model-to-mutexes.patch
bad/gpl_future-test.patch
bad/gregkh-debugfs_example.patch
bad/speakup-kconfig-fix.patch
bad/speakup-build-fix.patch
bad/pci-use-new-multi-phase-suspend-infrastructure.patch
bad/shot-accross-the-bow.patch
bad/no-more-non-gpl-modules.patch
bad/spi-device.patch
bad/ata_piix-multithread.patch
bad/uio-irq.patch
bad/pci-two-drivers-on-one-pci-device.patch
bad/pci-dynamic-id-cleanup.patch
bad/input-device.patch
bad/usb-stimulus.patch
driver/nozomi.patch
driver/kobject-put-kobject_actions-in-kobject.h.patch
driver/sysfs-implement-sysfs-manged-shadow-directory-support.patch
driver/sysfs-implement-sysfs_delete_link-and-sysfs_rename_link.patch
driver/sysfs-remove-first-pass-at-shadow-directory-support.patch
driver/driver-core-implement-shadow-directory-support-for-device-classes.patch
gregkh/gkh-version.patch
gregkh/sysfs-test.patch
gregkh/sysrq-u-laptop.patch
pci/pci_bridge-device.patch
pci/pci-piggy-bus.patch
pci/pci-move-prototypes-for-pci_bus_find_capability-to-include-linux-pci.h.patch
pci/pci-document-pci_iomap.patch
usb/usb-gotemp.patch

usb/kobject-put-kobject_actions-in-kobject.h.patch
usb/usb-add-the-concept-of-default-authorization-to-usb-hosts.patch
usb/usb-cleanup-usb_register_bus-and-hook-up-sysfs-group.patch
usb/usb-initialize-authorization-and-wusb-bits-in-usb-devices.patch
usb/usb-introduce-usb_device-authorization-bits.patch
usb/usb-usb_set_configuration-obeys-authorization.patch
usb/usb-usb.h-kernel-doc-additions.patch
HOWTO

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

---

Subject: Re: patch driver-core-implement-shadow-directory-support-for-device-classes.patch added to gregkh-2.
Posted by ebiederm on Sat, 21 Jul 2007 10:00:43 GMT
View Forum Message <> Reply to Message

<gregkh@suse.de> writes:

> This is a note to let you know that I've just added the patch titled
>
>   Subject: [PATCH 4/4] driver core: Implement shadow directory support for
> device classes.
>
> to my gregkh-2.6 tree. Its filename is
>
>   driver-core-implement-shadow-directory-support-for-device-classes.patch
>
> This tree can be found at
>   http://www.kernel.org/pub/linux/kernel/people/gregkh/gregkh-2.6/patches/

Hip hip hooray. The patches finally apply!!!!

Eric

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

---

Subject: Re: [PATCH 1/4] sysfs: Remove first pass at shadow directory support
Posted by Tejun Heo on Sun, 22 Jul 2007 18:35:27 GMT
View Forum Message <> Reply to Message

Hello,

Hope my late review isn't too late.

Eric W. Biederman wrote:
> While shadow directories appear to be a good idea, the current scheme
> of controlling their creation and destruction outside of sysfs appears
> to be a locking and maintenance nightmare in the face of sysfs
> directories dynamically coming and going.

Yeah, shadow directory was a real PITA while restructuring sysfs. I tried hard to keep it working but, without test case and detailed documentation, I'm pretty sure I broke something.

My feeling was that it was merged too early and implementation was too tightly coupled with other more regular paths. Anyways, glad shadow dirs are getting some loving care. If properly done, we should be able to simplify sysfs_get_dentry(), removal logic and save a pointer in sysfs_dirent.

> Which can now occur for
> directories containing network devices when CONFIG_SYSFS_DEPRECATED is
> not set.

Shadow directories were always pinned - both the shadowed one and shadows. Well, that was the theory at least.

> -int sysfs_rename_dir(struct kobject *kobj, struct sysfs_dirent *new_parent_sd,
> - const char *new_name)
> +int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
> {
> - struct sysfs_dirent *sd = kobj->sd;
> - struct dentry *new_parent = NULL;
> + 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;
> int error;
>
> + if (!kobj->parent)
> + return -EINVAL;

Please don't do this. The goal is to decouple sysfs and kobj.

> /* get dentries */
> + sd = kobj->sd;
> old_dentry = sysfs_get_dentry(sd);

>   if (IS_ERR(old_dentry)) {
>   error = PTR_ERR(old_dentry);
>   goto out_dput;
>   }
>
> - new_parent = sysfs_get_dentry(new_parent_sd);
> - if (IS_ERR(new_parent)) {
> - error = PTR_ERR(new_parent);
> + parent_sd = kobj->parent->sd;
> + parent = sysfs_get_dentry(parent_sd);
> + if (IS_ERR(parent)) {
> + error = PTR_ERR(parent);
>   goto out_dput;

You can simply do parent = old_dentry->d_parent. parent is guaranteed
to be there as long as we hold old_dentry. In the original code,
new_parent needed to be looked because new_parent wasn't necessarily
old_dentry's parent.

> @@ -965,10 +957,10 @@ int sysfs_rename_dir(struct kobject *kobj, struct sysfs_dirent
*new_parent_sd,
>   out_drop:
>   d_drop(new_dentry);
>   out_unlock:
> - mutex_unlock(&new_parent->d_inode->i_mutex);
> + mutex_unlock(&parent->d_inode->i_mutex);
>   out_dput:
>   kfree(dup_name);
> - dput(new_parent);
> + dput(parent);

So, dput(parent) can go away too.

> diff --git a/fs/sysfs/group.c b/fs/sysfs/group.c
> index f318b73..4606f7c 100644
> --- a/fs/sysfs/group.c
> +++ b/fs/sysfs/group.c
> @@ -13,7 +13,6 @@
> #include <linux/dcache.h>
> #include <linux/namei.h>
> #include <linux/err.h>
> -#include <linux/fs.h>
> #include <asm/semaphore.h>
> #include "sysfs.h"

Can you explain this one a bit?

Thanks.

--
tejun

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

---

Subject: Re: [PATCH 1/4] sysfs: Remove first pass at shadow directory support
Posted by ebiederm on Sun, 22 Jul 2007 19:17:18 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> Hello,
>
> Hope my late review isn't too late.
>
> Eric W. Biederman wrote:
>> While shadow directories appear to be a good idea, the current scheme
>> of controlling their creation and destruction outside of sysfs appears
>> to be a locking and maintenance nightmare in the face of sysfs
>> directories dynamically coming and going.
>
> Yeah, shadow directory was a real PITA while restructuring sysfs. I
> tried hard to keep it working but, without test case and detailed
> documentation, I'm pretty sure I broke something.
>
> My feeling was that it was merged too early and implementation was too
> tightly coupled with other more regular paths.

It's a problem because sysfs has this tendency especially before your
cleanups to be tightly coupled.

But yes the original implementation was probably factored at the wrong
level, not giving sysfs enough responsibility. Which became apparent
when the single objects started living all over the sysfs tree.

> Anyways, glad shadow
> dirs are getting some loving care. If properly done, we should be able
> to simplify sysfs_get_dentry(), removal logic and save a pointer in
> sysfs_dirent.

Hopefully. The logic in sysfs_resolve_for_create (i.e. which shadow
dir do you intend to create something is tricky with the current locking
logic).

>> Which can now occur for
>> directories containing network devices when CONFIG_SYSFS_DEPRECATED is
>> not set.
>
> Shadow directories were always pinned - both the shadowed one and
> shadows. Well, that was the theory at least.

Yes. The difference I was referring to was:
/sys/class/net/ used to be the only directory I needed to shadow.

Now there is one net/ in each networking device and several other
little things.

>> -int sysfs_rename_dir(struct kobject *kobj, struct sysfs_dirent
> *new_parent_sd,
>> -   const char *new_name)
>> +int sysfs_rename_dir(struct kobject * kobj, const char *new_name)
>> {
>> - struct sysfs_dirent *sd = kobj->sd;
>> - struct dentry *new_parent = NULL;
>> + 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;
>>   int error;
>>
>> + if (!kobj->parent)
>> + return -EINVAL;
>
> Please don't do this. The goal is to decouple sysfs and kobj.

I have no problem moving that test back to a higher level.
My practical goal of this first patch was to remove the original
shadow dir work so I had a clean slate to start with. I got
tired and perhaps didn't do as clean a break here as I could
have.

Do you mind an incremental patch to move the kobj->parent test?

As long as we don't break git-bisect support I would really prefer
to just build on top of the patches that are there.

It has been a major hair pulling exercise to get everything to get
everything to work in the presence of other parallel changes in sysfs.
I only had to relearn the locking and organization rules 3 times. I
really don't want to repeat it but I am happy to incrementally improve
things.

>>   /* get dentries */
>> + sd = kobj->sd;
>>   old_dentry = sysfs_get_dentry(sd);
>>   if (IS_ERR(old_dentry)) {
>>   error = PTR_ERR(old_dentry);
>>   goto out_dput;
>>   }
>>
>> - new_parent = sysfs_get_dentry(new_parent_sd);
>> - if (IS_ERR(new_parent)) {
>> - error = PTR_ERR(new_parent);
>> + parent_sd = kobj->parent->sd;
>> + parent = sysfs_get_dentry(parent_sd);
>> + if (IS_ERR(parent)) {
>> + error = PTR_ERR(parent);
>>   goto out_dput;
>
> You can simply do parent = old_dentry->d_parent. parent is guaranteed
> to be there as long as we hold old_dentry. In the original code,
> new_parent needed to be looked because new_parent wasn't necessarily
> old_dentry's parent.

Yes. This part is probably the ugliest, and least well factored part
of this patchset. sysfs_rename_dir wasn't something I could completely
revert to it's pre shadow directory state as that was no longer
available. What do you think of sysfs_rename_dir after the second patch in
the patchset?

I still think I have the limitation that new_parent and old_parent may
be separate. (Although clearly not at this point in the patch series).
I seem to remember trying to minimize the flux in sysfs_rename_dir and
eventually giving up. Having found a path that seemed to be good enough.

>> @@ -965,10 +957,10 @@ int sysfs_rename_dir(struct kobject *kobj, struct
> sysfs_dirent *new_parent_sd,
>>   out_drop:
>>   d_drop(new_dentry);
>>   out_unlock:
>> - mutex_unlock(&new_parent->d_inode->i_mutex);
>> + mutex_unlock(&parent->d_inode->i_mutex);
>>   out_dput:
>>   kfree(dup_name);
>> - dput(new_parent);
>> + dput(parent);
>
> So, dput(parent) can go away too.

I actually managed to remove sysfs_get_dentry entirely from
sysfs_rename_dir and to use sysfs_addrm_start.

>> diff --git a/fs/sysfs/group.c b/fs/sysfs/group.c
>> index f318b73..4606f7c 100644
>> --- a/fs/sysfs/group.c
>> +++ b/fs/sysfs/group.c
>> @@ -13,7 +13,6 @@
>> #include <linux/dcache.h>
>> #include <linux/namei.h>
>> #include <linux/err.h>
>> -#include <linux/fs.h>
>> #include <asm/semaphore.h>
>> #include "sysfs.h"
>
> Can you explain this one a bit?

This include was added by the original shadow directory support
and with it the original shadow directory support removed the include
became unnecessary. I forget the exact details.

But so I could catch subtle things like that is why my patchset is
structured as first a removal pass and then the new stuff.

Eric

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

---

Subject: Re: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.
Posted by Tejun Heo on Sun, 22 Jul 2007 19:47:21 GMT
View Forum Message <> Reply to Message

Hello,

Eric W. Biederman wrote:
> diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
> +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;

This is way too cryptic and, plus, no comment. This kind of stuff can
cause a lot of confusion later when other people wanna work on the
code. Please move s_name into sysfs_elem_* which need s_name and
create sysfs_elem_shadow which doesn't have ->name but has ->tag.

> +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;
> +}

Please don't use kobj inside sysfs. Use sysfs_dirent instead.

> @@ -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);

I think we can use some comment for subtle things like this.
DCACHE_UNHASHED is being tested without holding dcache_lock which also
can use some comment.

> @@ -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) {

sd->s_flags should be protected by sysfs_mutex. Please don't depend
on inode locking for synchronization internal to sysfs.

> +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);

Care to explain this a bit?

>   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)

This rename modification is painful. Please explain why we need to
rename nodes between shadows? Can't we just create new ones? Also,
please add some comment when performing black magic.

> @@ -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)

s_flags protection?

> @@ -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)

Please put this before sysfs_addrm_finish(). That's the only place
this function is used.

> +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;
> +}

Can we just add sd here and resolve the rest the same way as other nodes such that each shadow has its own dentry and inode? Am I missing something? Also, why do we need the intermediate shadowed sd at all? Can't we do the following?

* non-shadowed case

parent_sd - sd

* shadowed case

parent_sd - sd0
sd1
sd2

I think we can reduce considerable special case handlings if we do like the above including the implicit shadow creation, parent pruning and symlink tricks. After all, it's just multiple siblings sharing a name which needs some extra context to look up the correct one. We wouldn't even need 'shadow' at all.

Sorry but I don't think the current approach is the correct one. It's too painful and too much complexity is scattered all over the place. I'm afraid this implementation is going to be a maintenance nightmare.

--
tejun

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

---

Subject: Re: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.

Posted by gregkh on Sun, 22 Jul 2007 20:25:08 GMT
View Forum Message <> Reply to Message

On Mon, Jul 23, 2007 at 04:47:21AM +0900, Tejun Heo wrote:
> Sorry but I don't think the current approach is the correct one. It's
> too painful and too much complexity is scattered all over the place.
> I'm afraid this implementation is going to be a maintenance nightmare.

In looking over this again, and due to the fact that there are no

in-kernel users of this code, I'm going to drop the other patches in
this series and only keep the first one that removes the current
implementation.

Eric, is that ok? It gives you time to revisit these changes.

thanks,

greg k-h

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

---

Subject: Re: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.
Posted by ebiederm on Sun, 22 Jul 2007 22:07:32 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> Hello,
>
> Eric W. Biederman wrote:
>> diff --git a/fs/sysfs/dir.c b/fs/sysfs/dir.c
>> +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;
>
> This is way too cryptic and, plus, no comment. This kind of stuff can
> cause a lot of confusion later when other people wanna work on the
> code. Please move s_name into sysfs_elem_* which need s_name and
> create sysfs_elem_shadow which doesn't have ->name but has ->tag.

I'm been staring at this to long to know. I just know the name
is more or less reasonable and following how it is called tends
to show what it is used for.

In one sense the name is very much the tag. So I don't feel
bad about using it that way. In another sense if we can cleanup
the code by having the s_name field be a string that would

be an improvement and be appreciated.

>> +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;
>> +}
>
> Please don't use kobj inside sysfs. Use sysfs_dirent instead.

The interface to sysfs is kobjects and names, so I'm limited
in what I can do.

Where this is used I know a directory and a name I want to
remove. What I don't necessarily know is which shadow of that
directory I want to remove from without looking at the kobject.

sysfs_delete_link is a good code path to follow to understand
this.

>> @@ -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);
>
> I think we can use some comment for subtle things like this.
> DCACHE_UNHASHED is being tested without holding dcache_lock which also
> can use some comment.

Basically this is a test to see if we have already been placed in
the dcache. I'm trying to remember my reasoning

>> @@ -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) {
>
> sd->s_flags should be protected by sysfs_mutex. Please don't depend
> on inode locking for synchronization internal to sysfs.

Basically this is a set once bit. That is never expected to be cleared.
And is never expected to show up until it is set. So that is minor.

I was hoping to avoid taking the sysfs_mutex for non-shadow directories
while always having the same code.

Whatever moving the sysfs_mutex up just a little bit in that function
is trivial if it is important. As is potentially having two copies
of the directory operations.

>> +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);
>
> Care to explain this a bit?

Hmm. It looks like in all of the thrashing of sysfs I got my
tested goofed up. It should say:

```
 if (!(dir_sd->s_flags & SYSFS_FLAG_SHAODWED))
  sysfs_empty_dir(&acxt, dir_sd);
 else
  sysfs_remove_shadows(&acxt, dir_sd);
```

The logic is if this is an ordinary directory just empty it.
However if this directory has shadows empty each shadow.

Because we need to delete them all.

>>   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)
>
> This rename modification is painful. Please explain why we need to
> rename nodes between shadows? Can't we just create new ones? Also,
> please add some comment when performing black magic.

Well the context is more in the changelog. Although something in the
code might help.

Are you looking at just the diff or the applied patch?
It may just be that the diff looks horrible. I don't see
deep black magic in there that needs an explicit comment.

Please look at sysfs_rename_dir. While the shadow logic may be
overkill I think it would probably be worth moving rename dir in a
direction where we don't require the dentries to be in cache and
we can use sysfs_addrm_start in any event.

Which is the bulk of what I have done there.

>> @@ -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)
>
> s_flags protection?

sysfs_mutex? And the set once property.
Basically the check here is just a sanity check and I don't
think we will ever get there.

>> @@ -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)
>
> Please put this before sysfs_addrm_finish(). That's the only place
> this function is used.

Sure. I was also calling it in sysfs_move_dir but that case was
to much of a pain and I wasn't really using it so I removed it.

>> +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;
>> +}
>
> Can we just add sd here and resolve the rest the same way as other
> nodes such that each shadow has its own dentry and inode? Am I
> missing something? Also, why do we need the intermediate shadowed sd
> at all? Can't we do the following?

Sharing the struct inode means when we switch from one shadow to another
we keep the same inode lock. Which means we can resolve which shadow we
are working with inside of sysfs_addrm_start, because we don't have to
drop and reacquire the locks.

I do the hashed but not in the dcache dentry so that I don't confuse
the VFS with multiple dentries of the same name as children on the same
dentry. Currently they are locked in cache to simplify things.

>
> * non-shadowed case
>
> parent_sd - sd
>
> * shadowed case
>
> parent_sd - sd0
>   sd1
>   sd2
>
> I think we can reduce considerable special case handlings if we do
> like the above including the implicit shadow creation, parent pruning
> and symlink tricks. After all, it's just multiple siblings sharing a
> name which needs some extra context to look up the correct one. We
> wouldn't even need 'shadow' at all.

I disagree. Or at least I don't see what you are suggesting.
Currently I see sysfs_resolve_for_create and sysfs_resolve_for_remove
(the implicit shadow directory creation/finding) as fundamental
complexity of the problem. We may be able to simplify the
implementation of those to a small extent but I don't see those code
paths going away.

The big problem I have is I don't know which directories I will need

to shadow. Currently on my test system I have:
/sys/class/net
/sys/devices/pci0000:00/0000:00:1c.5/0000:04:00.0/net
/sys/devices/virtual/net

But hotpluging a new pci device could add yet another directory.
So the only code which actually has a clue which directories
I need to handle is the sysfs/kobject layer I can't handle it
externally. And adding to my fun my network namespaces (the tags)
are created and destroyed asynchronously to the devices coming
in and going. That is why I have the implicit creation, because
I don't know what is happening.

So while I think changing how exactly I use sysfs_dirents may
simplify some things I don't see things getting much simpler.

Originally things worked out more nicely because we had the
dcache in the form you suggest and the sysfs_dirent tree in the
current form. But we were always looking at the dcache when we
really cared. So things were a little nicer and there were one
or two fewer special cases. But the code was effectively the same.

Now things are a little worse because we don't use the dcache
tree for anything. So getting the full path name has a special
case.

What do I have to deal with.
- readdir knows it can return the inode number.
- readdir needs to only return one entry for a shadow.

- When I move a network device between namespaces I need sysfs
  to follow. In particular as I recall we have kobject attributes
  added by individual devices that I need to preserve. Which
  strongly suggests doing something a rename to switch contexts
  is what is needed.

  So for the rename I need to get sd->parent_sd is the old directory.

  Now perhaps I should be calling something besides device_rename
  while I move a network device between namespaces and I should have a
  completely different code path. But that doesn't feel correct to
  me.

- The dcache can only hold one entry with a given name so I need to
  use a magic follow_link to switch to the proper name.

- Implicit directory creation is essential because I don't know which
  directories I will be shadowing a priori.

> Sorry but I don't think the current approach is the correct one. It's
> too painful and too much complexity is scattered all over the place.
> I'm afraid this implementation is going to be a maintenance
> nightmare.

Frankly. The tightly coupled sysfs + kobject debacle of a user interface adds a lot of complexity to maintaining a stable interface to user space. And if something should pay it should be the sysfs code itself not the other pieces of the kernel that are stuck with sysfs. So I will happily argue that my interface to the upper levels is correct or very close to it.

At the same time. If we can find a way to do this with less complexity in sysfs I'm all for it. I just don't see it yet.

To some extent the ideal user space interface would be one where we have multiple different mounts of sysfs. At mount time each one calling shadow_ops->current_tag() and only displaying one tag in that sysfs mount instance. Given the current coupling of everything that still looks noticeably harder to implement then what I have done so far.

Eric

Subject: Re: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.
Posted by ebiederm on Sun, 22 Jul 2007 22:19:42 GMT
View Forum Message <> Reply to Message

Greg KH <gregkh@suse.de> writes:

> On Mon, Jul 23, 2007 at 04:47:21AM +0900, Tejun Heo wrote:
>> Sorry but I don't think the current approach is the correct one. It's
>> too painful and too much complexity is scattered all over the place.
>> I'm afraid this implementation is going to be a maintenance nightmare.
>
> In looking over this again, and due to the fact that there are no
> in-kernel users of this code, I'm going to drop the other patches in
> this series and only keep the first one that removes the current

> implementation.
>
> Eric, is that ok? It gives you time to revisit these changes.

Greg a big part of the reason I don't have internal kernel users
is keeping up with changes of sysfs has meant I haven't had time
to clean up and submit the patches for the users. With a little
luck I will have users by 2.6.24. I am in the process of cleaning
up those patches right now. So I can send the off to Dave Miller.

So please can we try and at least keep these sysfs patches in a
development tree so that people will see them.

Further while there are a few little nits I think mostly Tejun is
mostly objecting to the fundamental complexity of the problem rather
then to things that can be fixed by a cleaner implementation.

If it didn't take me a week every time I had to update this code
after Tejun changes the locking rules in fs/sysfs/dir.c or if there
was someone I could delegate the work of maintaining this code to
I probably would not mind dropping the patches for a little bit. As
it stands I am having horrible nightmares about how the internals
of sysfs will be completely different if you drop the last 3 patches
by the time I come back and I will need to spend several more weeks
just catching up.

Eric

Subject: Re: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.
Posted by Tejun Heo on Mon, 23 Jul 2007 03:52:25 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Further while there are a few little nits I think mostly Tejun is
> mostly objecting to the fundamental complexity of the problem rather
> then to things that can be fixed by a cleaner implementation.

Oh well, I don't think so but I might be wrong.

> If it didn't take me a week every time I had to update this code
> after Tejun changes the locking rules in fs/sysfs/dir.c or if there
> was someone I could delegate the work of maintaining this code to
> I probably would not mind dropping the patches for a little bit. As
> it stands I am having horrible nightmares about how the internals
> of sysfs will be completely different if you drop the last 3 patches
> by the time I come back and I will need to spend several more weeks
> just catching up.

Yeah, sysfs has gone through a lot of changes but I think most of
internal restructuring is complete now. What's left is removing kobj
completely from sysfs internals and interface.

We kind of share the pain here although yours seems much worse than
mine. Shadow directories have been major pain in the ass while
restructuring sysfs and I basically had to shoot in the dark because
there was no in-kernel user. I guess the blame falls on the timing.

I'll give a shot at the no intermediate shadowed directory
implementation. I think things will fit a lot easier that way but I
really dunno till I try. I'll try to post prototype early.

As long as the current shadow implementation doesn't get into mainline.
I'm okay with it staying in Greg's tree until this is resolved.

Thanks.

--
tejun

Subject: Re: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.
Posted by gregkh on Tue, 24 Jul 2007 07:26:19 GMT
View Forum Message <> Reply to Message

On Mon, Jul 23, 2007 at 12:52:25PM +0900, Tejun Heo wrote:
> Eric W. Biederman wrote:
> > Further while there are a few little nits I think mostly Tejun is
> > mostly objecting to the fundamental complexity of the problem rather
> > then to things that can be fixed by a cleaner implementation.
>
> Oh well, I don't think so but I might be wrong.

>
> > If it didn't take me a week every time I had to update this code
> > after Tejun changes the locking rules in fs/sysfs/dir.c or if there
> > was someone I could delegate the work of maintaining this code to
> > I probably would not mind dropping the patches for a little bit. As
> > it stands I am having horrible nightmares about how the internals
> > of sysfs will be completely different if you drop the last 3 patches
> > by the time I come back and I will need to spend several more weeks
> > just catching up.
>
> Yeah, sysfs has gone through a lot of changes but I think most of
> internal restructuring is complete now. What's left is removing kobj
> completely from sysfs internals and interface.
>
> We kind of share the pain here although yours seems much worse than
> mine. Shadow directories have been major pain in the ass while
> restructuring sysfs and I basically had to shoot in the dark because
> there was no in-kernel user. I guess the blame falls on the timing.
>
> I'll give a shot at the no intermediate shadowed directory
> implementation. I think things will fit a lot easier that way but I
> really dunno till I try. I'll try to post prototype early.
>
> As long as the current shadow implementation doesn't get into mainline.
> I'm okay with it staying in Greg's tree until this is resolved.

Don't worry, I will not be sending it on to Linus unless you give the ok
to do so :)

thanks,

greg k-h

Subject: Re: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.
Posted by Tejun Heo on Thu, 26 Jul 2007 08:00:05 GMT
View Forum Message <> Reply to Message

Hello,

Okay, some questions.

* What do you think about not allowing duplicate names across different

tags? ie. there's only one ethX anywhere but it's visible only in a
specific namespace (and maybe in the default global one). Or does
everyone need its own eth0. If this is acceptable, the problem becomes
much simpler.

* I think we can do away with the magic tag and use a pointer to
vfsmount instead. So, a process which wants to be in certain namespace
can bind-mount /sysfs to its own /sysfs and make needed sysfs nodes
bound to the mount. Does this sound okay? Such process should probably
be in its own chrooted environment to function properly.

* I haven't really followed the containers thread. Do people generally
agree on including it in mainline when we have all the fancy
virtualization stuff?

Thanks.

--
tejun

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

---

Subject: Re: Re: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.
Posted by dev on Fri, 27 Jul 2007 10:58:23 GMT
View Forum Message <> Reply to Message

Tejun Heo wrote:
> Hello,
>
> Okay, some questions.
>
> * What do you think about not allowing duplicate names across different
> tags? ie. there's only one ethX anywhere but it's visible only in a
> specific namespace (and maybe in the default global one). Or does
> everyone need its own eth0. If this is acceptable, the problem becomes
> _much_ simpler.

at least on checkpoint/restart names should be restored as is
and conflicts are unavoidable.
this includes ifindex as well by the way.

> * I think we can do away with the magic tag and use a pointer to
> vfsmount instead. So, a process which wants to be in certain namespace
> can bind-mount /sysfs to its own /sysfs and make needed sysfs nodes

> bound to the mount. Does this sound okay? Such process should probably
> be in its own chrooted environment to function properly.
>
> * I haven't really followed the containers thread. Do people generally
> agree on including it in mainline when we have all the fancy
> virtualization stuff?

Thanks,
Kirill

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

---

Subject: Re: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.
Posted by Carl-Daniel Hailfinge on Fri, 27 Jul 2007 20:59:30 GMT
View Forum Message <> Reply to Message

Hi,

On 26.07.2007 10:00, Tejun Heo wrote:
> Okay, some questions.
>
> * What do you think about not allowing duplicate names across different
> tags? ie. there's only one ethX anywhere but it's visible only in a
> specific namespace (and maybe in the default global one). Or does
> everyone need its own eth0. If this is acceptable, the problem becomes
> _much_ simpler.

Duplicating names across different namespaces (eth0 for everyone) is a feature which allows me to use a very similar configurations for different namespaces.
There's also a security argument saying that one namespace shouldn't be able to infer information about other namespaces. If an interface name must not be reused across namespaces you can enumerate the obvious interface name list from any namespace and find out which interface names are used by other namespaces. I'm not suggesting that this argument is valid, but it needs to be considered and if we decide it is invalid, we should document why.

Regards,
Carl-Daniel

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.
Posted by Tejun Heo on Mon, 30 Jul 2007 07:09:06 GMT
View Forum Message <> Reply to Message

Hello, Eric.

Tejun Heo wrote:
> Eric W. Biederman wrote:
>> Further while there are a few little nits I think mostly Tejun is
>> mostly objecting to the fundamental complexity of the problem rather
>> then to things that can be fixed by a cleaner implementation.
>
> Oh well, I don't think so but I might be wrong.

And I'm wrong. Mine didn't turn out to be much cleaner than yours.
What I did was (still broken)...

* No shadower/shadowee. Each dentry is tagged.
* dentries of tagged sd's are taken out of dcache and always go through
->lookup() where the correct sd is looked up considering the current tag.

Tagging and adding new entries could be done rather cleanly but shooting
down existing dentries on rename/move turned out to be a mess. Things
will be much simpler if no sysfs dentry is hashed on dcache and always
go through ->lookup() but that will hurt big machines.

The basic problem here is that dcache layer doesn't allow different
views and sysfs shadow is trying to work behind its back. I don't think
this is a viable approach. Both implementations bend too many rules and
are too fragile. It will be a genuine pain in the ass to maintain.

Sorry that I can't come up with an alternative but NACK.

--
tejun

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

Subject: Re: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.

Tejun Heo wrote:
> Hello, Eric.
>
> Tejun Heo wrote:
>
>>Eric W. Biederman wrote:
>>
>>>Further while there are a few little nits I think mostly Tejun is
>>>mostly objecting to the fundamental complexity of the problem rather
>>>then to things that can be fixed by a cleaner implementation.
>>
>>Oh well, I don't think so but I might be wrong.
>
>
> And I'm wrong. Mine didn't turn out to be much cleaner than yours.
> What I did was (still broken)...
>
> * No shadower/shadowee. Each dentry is tagged.
> * dentries of tagged sd's are taken out of dcache and always go through
> ->lookup() where the correct sd is looked up considering the current tag.
>
> Tagging and adding new entries could be done rather cleanly but shooting
> down existing dentries on rename/move turned out to be a mess. Things
> will be much simpler if no sysfs dentry is hashed on dcache and always
> go through ->lookup() but that will hurt big machines.
>
> The basic problem here is that dcache layer doesn't allow different
> views and sysfs shadow is trying to work behind its back. I don't think
> this is a viable approach. Both implementations bend too many rules and
> are too fragile. It will be a genuine pain in the ass to maintain.
>
> Sorry that I can't come up with an alternative but NACK.

Imho then OpenOVZ approach with multiple sysfs trees is better.
it allows to use cached dentries with moultiple sysfs mounts
each having different view.
It also allows to hide hw-related entries and events from the containers
and has quite little modifications in the code.

Thanks,
Kirill

Subject: Re: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.
Posted by Tejun Heo on Mon, 30 Jul 2007 13:06:22 GMT
View Forum Message <> Reply to Message

Hello, Kirill.

Kirill Korotaev wrote:
> Imho then OpenOVZ approach with multiple sysfs trees is better.
> it allows to use cached dentries with moultiple sysfs mounts
> each having different view.
> It also allows to hide hw-related entries and events from the containers
> and has quite little modifications in the code.

I thought something like supermount plus some twists or fuse based sysfs proxy would fit better. Dunno whether or how uevent and polling stuff can work that way tho. Note that sysfs no longer keeps dentries and inodes pinned. It might make the shared dentry stuff harder.

Thanks.

--
tejun

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

Subject: Re: [PATCH 2/4] sysfs: Implement sysfs manged shadow directory support.
Posted by ebiederm on Mon, 30 Jul 2007 15:36:48 GMT
View Forum Message <> Reply to Message

Tejun Heo <htejun@gmail.com> writes:

> Hello,
>
> Okay, some questions.
>
> * What do you think about not allowing duplicate names across different
> tags? ie. there's only one ethX anywhere but it's visible only in a
> specific namespace (and maybe in the default global one). Or does
> everyone need its own eth0. If this is acceptable, the problem becomes
> _much_ simpler.

I agree, and unfortunately this is the problem I am really trying to solve.
Everyone namespace has it's own loopback interface.

> * I think we can do away with the magic tag and use a pointer to
> vfsmount instead. So, a process which wants to be in certain namespace
> can bind-mount /sysfs to its own /sysfs and make needed sysfs nodes
> bound to the mount. Does this sound okay? Such process should probably
> be in its own chrooted environment to function properly.

That would actually be preferable.

> * I haven't really followed the containers thread. Do people generally
> agree on including it in mainline when we have all the fancy
> virtualization stuff?

Generally. Assuming all of the i's are dotted and all of the t's crossed.

Eric

_____