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

_____

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