Subject: Re: [PATCH 06/11] sysfs: Implement sysfs tagged directory support.
Posted by ebiederm on Thu, 26 Jun 2008 20:21:24 GMT
View Forum Message <> Reply to Message

Tejun thank you for the review, and my apologies for the delayed
reply.

Tejun Heo <htejun@gmail.com> writes:

>> Index: linux-mm/include/linux/sysfs.h
>> ===
>> --- linux-mm.orig/include/linux/sysfs.h
>> +++ linux-mm/include/linux/sysfs.h
>> @@ -80,6 +80,14 @@ struct sysfs_ops {
>> ssize_t (*store)(struct kobject *,struct attribute *,const char *, size_t);
>> };
>>
>> +struct sysfs_tag_info {
>> +};
>> +
>> +struct sysfs_tagged_dir_operations {
>> + const void *(*sb_tag)(struct sysfs_tag_info *info);
>> + const void *(*kobject_tag)(struct kobject *kobj);
>> +};
>
> As before, I can't bring myself to like this interface. Is computing
> tags dynamically really necessary? Can't we do the followings?

It isn't so much computing tags dynamically but rather it is reading them
from where they are stored.

> tag = sysfs_allocate_tag(s);
> sysfs_enable_tag(kobj (or sd), tag);
> sysfs_sb_show_tag(sb, tag);
>
> Where tags are allocated using ida and each sb has bitmap of enabled
> tags so that sysfs ops can simply use something like the following to
> test whether it's enabled.
>
> bool sysfs_tag_enabled(sb, tag)
> {
> return sysfs_info(sb)->tag_map & (1 << tag);
> }

Youch that seems limiting. The expectation is that we could have
as many as 100 different containers in use on a single system at one
time. So 100 apparent copies of the network stack.

There is also a second dimension here we multiplex different
directories based on different sets of tags. One directory based
on user namespaces another on the network namespaces.

The tags in practice are just pointers to the namespace pointers.

So while we could use the ida technique to specify which set of tags
we are talking about for a directory it isn't sufficient.

The question sysfs_tag_enabled(sb, tag) makes no sense to me.
Especially in the context of needed a sysfs_sb_show_tag(sb, tag);

The current structure is because of all of the darn fool races and
magic that sysfs does. We have to say for a given directory: Your
contents will always be tagged, and only those that one tag that
matches what was captured by the superblock when sysfs is mounted
will be shown.

> Tags which can change dynamically seems too confusing to me and it
> makes things difficult to verify as it's unclear how those tags are
> gonna to change.

We have a fundamental issue that we have to handle, and it sounds like
you are proposing something that will not handle it.

- network devices can move between namespaces.
- network devices have driver specific sysfs attributes hanging off of them.

So we have to move the network devices and their sysfs attributes
between namespaces, and I implemented that in kobject_rename,
sysfs_rename path.

The tags on a kobject can only change during a rename operation.
So when the change happens is well defined. Further there is a
set of functions: sysfs_creation_tag, sysfs_removal_tag,
sysfs_lookup_tag, sysfs_dirent_tag which makes it clear what we
are doing.

If you really don't like how the tags are managed we need to talk
about how we store the tags on kobjects and on the super block.

Registering a set of tags could easily make the sb_tag function
obsolete, and that is one small piece of code so it is no big deal.

```
struct sysfs_tag_type_operations {
 const void *(*mount_tag)(void);
       const void *(*kobject_tag)(struct kobject *kobj);
```

};

Then we could do:
struct sysfs_sbtag_operations *tag_type_ops[MAX_TAG_TYPES];

And sysfs_tag_info could become.
struct sysfs_tag_info {
	void *tag[MAX_TAG_TYPES];
};

During subsystem initialization we could call
tag_type = sysfs_allocate_tag_type();

Just after the subsystem creates a directory.
sysfs_enable_tagging(kobj/sd, tag_type);

Then anytime we currently call sb_tag during lookup we can instead just look at sysfs_info(sb)->tag[tag_type] and compare that with sd->s_tag.tag.

The actual tag values themselves are current stored in the object in which the kobject is embedded.

So we still need to call kobject_tag when we create or rename something in a tagged directory. So we know what the tag is.

When we go to remove a kobj using the existing tag on the object is the right choice.

Rename is the fun case where we need to grab the old tag from the sd and place on it the new tag from kobject_tag.

One of the big problems at least with the class directories is that the lifetimes are completely decoupled the between the tags and the subsystem objects and subsystem directories that need to be tagged. This isn't a set things up at the start of your subsystem and everything is happy situation. To handle the races there must be support at least at the kobject level for handling this in the network namespace case.

Eric