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