

[RFC][PATCH 1 of 4] Configfs is really sysfs

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-08/7749.html>

From: Daniel Phillips (phillips_at_istop.com)

Date: 08/31/05

To: linux-kernel@vger.kernel.org, Andrew Morton <akpm@osdl.org>

Date: Wed, 31 Aug 2005 08:54:39 +1000

Hi Andrew,

Configfs blithely ingests `kobject.h` and `kobject.c` into itself, just changing the names. Furthermore, more than half of configfs is copied verbatim from sysfs, the only difference being the name changes. After undoing the name changes and adding a few new fields to `kobject` structures, configfs is able to use the real thing instead of its own imitation.

The changes I made to `kobject.h` and `sysfs.h` are:

- * add module owner to `kobj_type`.
- * add `group_operations` to `kobj_type` (because configfs does it this way not because it is right)
- * add a `children` field to `kset`. This is likely the same as the blandly named "list" field but I haven't confirmed it.
- * add a `default_groups` field to `kset`, analogous to the `default_attrs` of `kobj_type`. Hmm, somebody seems to be mixing up types and containers here, but let's just close our eyes for now.
- * add an `s_links` field to `sysfs_dirent` to support configfs's user createable symlinks.
- * add two new methods to `sysfs_ops` for fancy symlink hooks
- * add a questionable release method to `sysfs_ops`. Sysfs and configfs have slightly different notions of when to release objects, one of them is probably wrong.

That's it, no new fields in `kobjects` themselves, and just three or four fields in other allocateable structures. After these changes, no structures at all are left in `configfs.h`. Configfs is now running happily using the `kobject` machinery instead of its own mutated clones and unsurprisingly, sysfs still runs happily too. These changes are all found in the first patch of this series.

Linux-Kernel: [RFC][PATCH 1 of 4] Configfs is really sysfs

I then looked into exactly how configfs and sysfs are different. To reduce the noise, I concatenated all the files in each directory into two single files. With redundant declarations removed, configfs came in at 1897 lines and sysfs at 1680. Diffing those two files shows:

```
diff -u fs/sysfs/sysfs.c fs/configfs/configfs.c | diffstat configfs.c | 1497
```

```
+++++-----  
1 files changed, 857 insertions(+), 640 deletions(-)
```

So we see that two thirds of sysfs made it into configfs unchanged. Of the remaining one third that configfs has not copied, about one third supports read/write/mmapable attribute files (why should configfs not have them too?), a little less than a third involves needlessly importing its own version of setattr, and the remainder, about 300 lines, exports the kernel interface for manipulating the user-visible sysfs tree.

Allowing for a few lines of fluff, configfs's value add is about 750 lines of user space glue for namespace operations. Nothing below that glue layer is changed, except cosmetically. So configfs really is sysfs. By adding about 300 lines to configfs we can add the vfs-bypass code, and voila, configfs becomes sysfs. Another 200 lines gives us the binary blob attributes as well. There is no reason whatsoever for configfs and sysfs to live on as separate code bases. If we really want to make a distinction, we can make the distinction with a flag.

But it would be stupid to forbid users from creating directories in sysfs or to forbid kernel modules from directly tweaking a configfs namespace. Why should the kernel not be able to add objects to a directory a user created? It should be up to the module author to decide these things.

Please do not push configfs to stable in this form. It is not actually a new filesystem, it is an extension to sysfs. Merging it as is would add more than a thousand lines of pointless kernel bloat. If indeed we wish to present exactly the semantics configfs now offers, we do not need a separate code base to do so.

The four patches in this patch set:

- 1) Add new fields to kobjects; update other headers to match
- 2) Sysfs all in one file
- 3) Configfs all in one file
- 4) A configfs kernel example using sysfs instead of configfs structures

Regards,

Daniel

```
diff -up --recursive 2.6.13-rc5-mm1.clean/include/linux/configfs.h  
2.6.13-rc5-mm1/include/linux/configfs.h  
--- 2.6.13-rc5-mm1.clean/include/linux/configfs.h 2005-08-09 18:23:31.000000000 -0400  
+++ 2.6.13-rc5-mm1/include/linux/configfs.h 2005-08-29 18:30:41.000000000 -0400
```

@@ -46,120 +46,32 @@

```

#define CONFIGFS_ITEM_NAME_LEN 20

-struct module;
-
-struct configfs_item_operations;
-struct configfs_group_operations;
-struct configfs_attribute;
-struct configfs_subsystem;
-
-struct config_item {
- char *ci_name;
- char ci_namebuf[CONFIGFS_ITEM_NAME_LEN];
- struct kref ci_kref;
- struct list_head ci_entry;
- struct config_item *ci_parent;
- struct config_group *ci_group;
- struct config_item_type *ci_type;
- struct dentry *ci_dentry;
-};
-
-extern int config_item_set_name(struct config_item *, const char *, ...);
-
-static inline char *config_item_name(struct config_item * item)
-{
- return item->ci_name;
-}
-
-extern void config_item_init(struct config_item *);
-extern void config_item_init_type_name(struct config_item *item,
- const char *name,
- struct config_item_type *type);
-extern void config_item_cleanup(struct config_item *);
-
-extern struct config_item * config_item_get(struct config_item *);
-extern void config_item_put(struct config_item *);
-
-struct config_item_type {
- struct module *ct_owner;
- struct configfs_item_operations *ct_item_ops;
- struct configfs_group_operations *ct_group_ops;
- struct configfs_attribute **ct_attrs;
-};
-
+extern void kobject_init_type_name(struct kobject *item, const char *name, struct kobj_type *type);

/**
- * group - a group of config_items of a specific type, belonging
+ * group - a group of kobjects of a specific type, belonging
 * to a specific subsystem.

```

Linux-Kernel: [RFC][PATCH 1 of 4] Configfs is really sysfs

```
*/

-struct config_group {
- struct config_item cg_item;
- struct list_head cg_children;
- struct configfs_subsystem *cg_subsys;
- struct config_group **default_groups;
-};
-
+extern void config_group_init(struct kset *group);
+extern void config_group_init_type_name(struct kset *group, const char *name, struct kobj_type *type);

-extern void config_group_init(struct config_group *group);
-extern void config_group_init_type_name(struct config_group *group,
- const char *name,
- struct config_item_type *type);
-
-
-static inline struct config_group *to_config_group(struct config_item *item)
+static inline struct kset *to_config_group(struct kobject *item)
{
- return item ? container_of(item,struct config_group,cg_item) : NULL;
+ return item ? container_of(item,struct kset,kobj) : NULL;
}

-static inline struct config_group *config_group_get(struct config_group *group)
+static inline struct kset *config_group_get(struct kset *group)
{
- return group ? to_config_group(config_item_get(&group->cg_item)) : NULL;
+ return group ? to_config_group(kobject_get(&group->kobj)) : NULL;
}

-static inline void config_group_put(struct config_group *group)
+static inline void config_group_put(struct kset *group)
{
- config_item_put(&group->cg_item);
+ kobject_put(&group->kobj);
}

-extern struct config_item *config_group_find_obj(struct config_group *, const char *);
-
-
-struct configfs_attribute {
- char *ca_name;
- struct module *ca_owner;
- mode_t ca_mode;
-};
-
-
-/*
- * If allow_link() exists, the item can symlink(2) out to other
```

Linux-Kernel: [RFC][PATCH 1 of 4] Configfs is really sysfs

```
- * items. If the item is a group, it may support mkdir(2).
- * Groups supply one of make_group() and make_item(). If the
- * group supports make_group(), one can create group children. If it
- * supports make_item(), one can create config_item children. If it has
- * default_groups on group->default_groups, it has automatically created
- * group children. default_groups may coexist alongside make_group() or
- * make_item(), but if the group wishes to have only default_groups
- * children (disallowing mkdir(2)), it need not provide either function.
- * If the group has commit(), it supports pending and committed (active)
- * items.
- */
-struct configfs_item_operations {
- void (*release)(struct config_item *);
- ssize_t (*show_attribute)(struct config_item *, struct configfs_attribute *, char *);
- ssize_t (*store_attribute)(struct config_item *, struct configfs_attribute *, const char *, size_t);
- int (*allow_link)(struct config_item *src, struct config_item *target);
- int (*drop_link)(struct config_item *src, struct config_item *target);
-};
-
-struct configfs_group_operations {
- struct config_item *(*make_item)(struct config_group *group, const char *name);
- struct config_group *(*make_group)(struct config_group *group, const char *name);
- int (*commit_item)(struct config_item *item);
- void (*drop_item)(struct config_group *group, struct config_item *item);
-};
-
+extern struct kobject *config_group_find_obj(struct kset *, const char *);

/**
@@ -185,20 +97,13 @@ struct configfs_group_operations {
#endif

-struct configfs_subsystem {
- struct config_group su_group;
- struct semaphore su_sem;
-};
-
-static inline struct configfs_subsystem *to_configfs_subsystem(struct config_group *group)
+static inline struct subsystem *to_configfs_subsystem(struct kset *kset)
{
- return group ?
- container_of(group, struct configfs_subsystem, su_group) :
- NULL;
+ return kset ? container_of(kset, struct subsystem, kset) : NULL;
}

-int configfs_register_subsystem(struct configfs_subsystem *subsys);
-void configfs_unregister_subsystem(struct configfs_subsystem *subsys);
+int configfs_register_subsystem(struct subsystem *subsys);
```

Linux-Kernel: [RFC][PATCH 1 of 4] Configfs is really sysfs

```

+void configfs_unregister_subsystem(struct subsystem *subsys);

#endif /* __KERNEL__ */

diff -up --recursive 2.6.13-rc5-mm1.clean/include/linux/kobject.h 2.6.13-rc5-mm1/include/linux/kobject.h
--- 2.6.13-rc5-mm1.clean/include/linux/kobject.h 2005-08-09 18:23:13.000000000 -0400
+++ 2.6.13-rc5-mm1/include/linux/kobject.h 2005-08-29 02:40:27.000000000 -0400
@@ -69,7 +69,9 @@ extern char * kobject_get_path(struct ko

struct kobj_type {
    void (*release)(struct kobject *);
+ struct module *ct_owner;
    struct sysfs_ops * sysfs_ops;
+ struct configfs_group_operations *ct_group_ops;
    struct attribute ** default_attrs;
};

@@ -106,6 +108,10 @@ struct kset {
    spinlock_t list_lock;
    struct kobject kobj;
    struct kset_hotplug_ops * hotplug_ops;
+
+ /* configfs fields: merge us! */
+ struct list_head cg_children;
+ struct kset **default_groups;
};

@@ -144,7 +150,7 @@ extern struct kobject * kset_find_obj(st
 * Use this when initializing an embedded kset with no other
 * fields to initialize.
 */
-#define set_kset_name(str) .kset = { .kobj = { .name = str } }
+#define set_kset_name(str) .kset = { .kobj = { .k_name = str } }

@@ -156,7 +162,7 @@ struct subsystem {
#define decl_subsys(_name, _type, _hotplug_ops) \
struct subsystem _name##_subsys = { \
    .kset = { \
- .kobj = { .name = __stringify(_name) }, \
+ .kobj = { .k_name = __stringify(_name) }, \
    .ktype = _type, \
    .hotplug_ops = _hotplug_ops, \
    } \
@@ -164,7 +170,7 @@ struct subsystem _name##_subsys = { \
#define decl_subsys_name(_varname, _name, _type, _hotplug_ops) \
struct subsystem _varname##_subsys = { \
    .kset = { \
- .kobj = { .name = __stringify(_name) }, \

```

Linux-Kernel: [RFC][PATCH 1 of 4] Configfs is really sysfs

```
+ .kobj = { .k_name = __stringify(_name) }, \
    .ktype = _type, \
    .hotplug_ops = _hotplug_ops, \
} \
diff -up ---recursive 2.6.13-rc5-mm1.clean/include/linux/sysfs.h 2.6.13-rc5-mm1/include/linux/sysfs.h
--- 2.6.13-rc5-mm1.clean/include/linux/sysfs.h 2005-08-09 18:23:13.000000000 -0400
+++ 2.6.13-rc5-mm1/include/linux/sysfs.h 2005-08-29 16:58:02.000000000 -0400
@@ -12,6 +12,7 @@

#include <asm/atomic.h>

+struct kset;
struct kobject;
struct module;

@@ -63,12 +64,36 @@ struct bin_attribute {
struct sysfs_ops {
    ssize_t (*show)(struct kobject *, struct attribute *,char *);
    ssize_t (*store)(struct kobject *,struct attribute *,const char *, size_t);
+ int (*allow_link)(struct kobject *src, struct kobject *target);
+ int (*drop_link)(struct kobject *src, struct kobject *target);
+ void (*release)(struct kobject *);
+};
+
+/*
+ * If allow_link() exists, the item can symlink(2) out to other
+ * items. If the item is a group, it may support mkdir(2).
+ * Groups supply one of make_group() and make_item(). If the
+ * group supports make_group(), one can create group children. If it
+ * supports make_item(), one can create kobject children. If it has
+ * default_groups on group->default_groups, it has automatically created
+ * group children. default_groups may coexist alongsize make_group() or
+ * make_item(), but if the group wishes to have only default_groups
+ * children (disallowing mkdir(2)), it need not provide either function.
+ * If the group has commit(), it supports pending and committed (active)
+ * items.
+ */
+struct configfs_group_operations {
+ struct kobject *(*make_item)(struct kset *group, const char *name);
+ struct kset *(*make_group)(struct kset *group, const char *name);
+ int (*commit_item)(struct kobject *item);
+ void (*drop_item)(struct kset *group, struct kobject *item);
+};

struct sysfs_dirent {
    atomic_t s_count;
    struct list_head s_sibling;
    struct list_head s_children;
+ struct list_head s_links; /* configfs */
    void * s_element;
    int s_type;

```

Linux-Kernel: [RFC][PATCH 1 of 4] Configfs is really sysfs

```
umode_t s_mode;
diff -up --recursive 2.6.13-rc5-mm1.clean/lib/kobject.c 2.6.13-rc5-mm1/lib/kobject.c
--- 2.6.13-rc5-mm1.clean/lib/kobject.c 2005-08-09 18:23:13.000000000 -0400
+++ 2.6.13-rc5-mm1/lib/kobject.c 2005-08-29 21:29:24.000000000 -0400
@@ -344,7 +344,9 @@ void kobject_cleanup(struct kobject * ko
     kfree(kobj->k_name);
     kobj->k_name = NULL;
     if (t && t->release)
- t->release(kobj);
+ t->release(kobj); /* which one of us... */
+ if (t && t->sysfs_ops && t->sysfs_ops->release)
+ t->sysfs_ops->release(kobj); /* ...is bogus? */
     if (s)
         kset_put(s);
     if (parent)
-

```

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@vger.kernel.org

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>