
Subject: [PATCH 2/4] The character devices layer changes
Posted by [Pavel Emelianov](#) on Tue, 08 Jan 2008 09:12:51 GMT
[View Forum Message](#) <> [Reply to Message](#)

These changes include the API for the control group to map/remap/unmap the devices with their permissions and one important thing.

The fact is that the struct cdev is cached in the inode for faster access, so once we looked one up we go through the fast path and omit the kobj_lookup() call. This is no longer good when we restrict the access to cdevs.

To address this issue, I store the last_perm and last(_map) fields on the struct cdev (and protect them with the cdev_lock) and force the re-lookup in the kobj mappings if needed.

I know, this might be slow, but I have two points for it:

1. The re-lookup happens on open() only which is not a fast-path. Besides, this is so for block layer and nobody complains;
2. On a well-isolated setup, when each container has its own filesystem this is no longer a problem - each cgroup will cache the cdev on its inode and work good.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/fs/char_dev.c b/fs/char_dev.c
index c3bfa76..2b821ef 100644
--- a/fs/char_dev.c
+++ b/fs/char_dev.c
@@ -22,6 +22,8 @@
#include <linux/mutex.h>
#include <linux/backing-dev.h>

+#include <linux/devscontrol.h>
+
#ifdef CONFIG_KMOD
#include <linux/kmod.h>
#endif
@@ -362,17 +364,25 @@ int chrdev_open(struct inode * inode, struct file * filp)
    struct cdev *p;
    struct cdev *new = NULL;
    int ret = 0;
+ struct kobj_map *map;
+ mode_t mode;
```

```

+
+ map = task_cdev_map(current);
+ if (map == NULL)
+ map = cdev_map;

    spin_lock(&cdev_lock);
    p = inode->i_cdev;
- if (!p) {
+ if (!p || p->last != map) {
    struct kobject *kobj;
    int idx;
+
    spin_unlock(&cdev_lock);
- kobj = kobj_lookup(cdev_map, inode->i_rdev, &idx);
+ kobj = kobj_lookup(map, inode->i_rdev, &mode, &idx);
    if (!kobj)
        return -ENXIO;
    new = container_of(kobj, struct cdev, kobj);
+ BUG_ON(p != NULL && p != new);
    spin_lock(&cdev_lock);
    p = inode->i_cdev;
    if (!p) {
@@ -382,12 +392,24 @@ int chrdev_open(struct inode * inode, struct file * filp)
        new = NULL;
    } else if (!cdev_get(p))
        ret = -ENXIO;
+ else {
+ p->last = map;
+ p->last_mode = mode;
+ }
    } else if (!cdev_get(p))
        ret = -ENXIO;
+ else
+ mode = p->last_mode;
    spin_unlock(&cdev_lock);
    cdev_put(new);
    if (ret)
        return ret;
+
+ if ((filp->f_mode & mode) != filp->f_mode) {
+ cdev_put(p);
+ return -EACCES;
+ }
+
    filp->f_op = fops_get(p->ops);
    if (!filp->f_op) {
        cdev_put(p);
@@ -461,6 +483,64 @@ int cdev_add(struct cdev *p, dev_t dev, unsigned count)

```

```

    return kobj_map(cdev_map, dev, count, NULL, exact_match, exact_lock, p);
}

#ifdef CONFIG_CGROUP_DEVS
+static inline void cdev_map_reset(struct kobj_map *map, struct cdev *c)
+{
+ spin_lock(&cdev_lock);
+ if (c->last == map)
+  c->last = NULL;
+ spin_unlock(&cdev_lock);
+}
+
+int cdev_add_to_map(struct kobj_map *map, dev_t dev, int all, mode_t mode)
+{
+ int tmp;
+ struct kobject *k;
+ struct cdev *c;
+
+ k = kobj_lookup(cdev_map, dev, NULL, &tmp);
+ if (k == NULL)
+  return -ENODEV;
+
+ c = container_of(k, struct cdev, kobj);
+ tmp = kobj_remap(map, dev, mode, all ? MINORMASK : 1, NULL,
+  exact_match, exact_lock, c);
+ if (tmp < 0) {
+  cdev_put(c);
+  return tmp;
+ }
+
+ cdev_map_reset(map, c);
+ return 0;
+}
+
+int cdev_del_from_map(struct kobj_map *map, dev_t dev, int all)
+{
+ int tmp;
+ struct kobject *k;
+ struct cdev *c;
+
+ k = kobj_lookup(cdev_map, dev, NULL, &tmp);
+ if (k == NULL)
+  return -ENODEV;
+
+ c = container_of(k, struct cdev, kobj);
+ kobj_unmap(map, dev, all ? MINORMASK : 1);
+
+ cdev_map_reset(map, c);

```

```

+
+ cdev_put(c);
+ cdev_put(c);
+ return 0;
+}
+
+void cdev_iterate_map(struct kobj_map *map,
+ int (*fn)(dev_t, int, mode_t, void *), void *x)
+{
+ kobj_map_iterate(map, fn, x);
+}
+
+
+static void cdev_unmap(dev_t dev, unsigned count)
+{
+ kobj_unmap(cdev_map, dev, count);
@@ -542,9 +622,19 @@ static struct kobject *base_probe(dev_t dev, int *part, void *data)
+ return NULL;
+}

+struct kobj_map *cdev_map_init(void)
+{
+ return kobj_map_init(base_probe, &chrdevs_lock);
+}
+
+void cdev_map_fini(struct kobj_map *map)
+{
+ kobj_map_fini(map);
+}
+
+void __init chrdev_init(void)
+{
+ cdev_map = kobj_map_init(base_probe, &chrdevs_lock);
+ cdev_map = cdev_map_init();
+ bdi_init(&directly_mappable_cdev_bdi);
+}

diff --git a/include/linux/cdev.h b/include/linux/cdev.h
index 1e29b13..d72a2a1 100644
--- a/include/linux/cdev.h
+++ b/include/linux/cdev.h
@@ -9,6 +9,7 @@
 struct file_operations;
 struct inode;
 struct module;
+struct kobj_map;

 struct cdev {

```

```

    struct kobject kobj;
@@ -17,6 +18,8 @@ struct cdev {
    struct list_head list;
    dev_t dev;
    unsigned int count;
+ struct kobj_map *last;
+ mode_t last_mode;
};

void cdev_init(struct cdev *, const struct file_operations *);
@@ -33,5 +36,11 @@ void cd_forget(struct inode *);

extern struct backing_dev_info directly_mappable_cdev_bdi;

+int cdev_add_to_map(struct kobj_map *map, dev_t dev, int all, mode_t mode);
+int cdev_del_from_map(struct kobj_map *map, dev_t dev, int all);
+struct kobj_map *cdev_map_init(void);
+void cdev_map_fini(struct kobj_map *map);
+void cdev_iterate_map(struct kobj_map *,
+ int (*fn)(dev_t, int, mode_t, void *), void *);
#endif
#endif

```

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