Subject: [PATCH] Filesystems visibility control group
Posted by Pavel Emelianov on Thu, 07 Feb 2008 15:04:52 GMT
View Forum Message <> Reply to Message

After making the devices visibility cg, I thought, that it might
be useful to have a filesystem visibility control group.

The patch is rather simple - all the code is in fs/filesystems.c

The main idea is in file_system_proxy object. This coincides in
its three first fields with file_system_type (name, flags and next)
and is used to lookup the file system. To distinguish between
them I use a FS_IS_PROXY flag.

Having this proxy is the easiest way to keep the global list and
file_system_type structure (almost) untouched and simplify the code.

The filesystems.list file syntax is simple: [+-]<name> without
a '\n' at the end. Made for 2.6.24-rc8-mm1

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

---

```
diff --git a/fs/filesystems.c b/fs/filesystems.c
index f37f872..b8189a4 100644
--- a/fs/filesystems.c
+++ b/fs/filesystems.c
@@ -27,9 +27,214 @@
  * Once the reference is obtained we can drop the spinlock.
  */

+/*
+ * the first three fields of it must coincide with the
+ * struct file_system_type
+ */
+
+struct file_system_proxy {
+ char *name;
+ struct file_system_proxy *next;
+ int fs_flags;
+ struct file_system_type *fs;
+};
+
 static struct file_system_type *file_systems;
 static DEFINE_RWLOCK(file_systems_lock);

+#ifdef CONFIG_CGROUP_FS
```

```
+#include <linux/cgroup.h>
+
+struct fs_cgroup {
+ struct cgroup_subsys_state css;
+
+ struct file_system_proxy *fs;
+};
+
+static inline struct fs_cgroup *css_to_fs(struct cgroup_subsys_state *ss)
+{
+ return container_of(ss, struct fs_cgroup, css);
+}
+
+static inline struct fs_cgroup *cgroup_to_fs(struct cgroup *c)
+{
+ return css_to_fs(cgroup_subsys_state(c, fs_subsys_id));
+}
+
+static inline struct file_system_type *task_fs_types(struct task_struct *t)
+{
+ struct cgroup_subsys_state *css;
+
+ css = task_subsys_state(t, fs_subsys_id);
+ if (css->cgroup->parent == NULL)
+  return file_systems;
+ else
+  return (struct file_system_type *)css_to_fs(css)->fs;
+}
+
+static struct cgroup_subsys_state *
+fs_create(struct cgroup_subsys *ss, struct cgroup *cont)
+{
+ struct fs_cgroup *fs;
+
+ fs = kzalloc(sizeof(struct fs_cgroup), GFP_KERNEL);
+ if (fs == NULL)
+  return ERR_PTR(-ENOMEM);
+ else
+  return &fs->css;
+}
+
+static void fs_destroy(struct cgroup_subsys *ss, struct cgroup *cont)
+{
+ struct fs_cgroup *fcg;
+ struct file_system_proxy *p, *next;
+
+ fcg = cgroup_to_fs(cont);
+
```

```
+ write_lock(&file_systems_lock);
+ for (p = fcg->fs; p != NULL; p = next) {
+  next = p->next;
+  put_filesystem(p->fs);
+  kfree(p);
+ }
+ write_unlock(&file_systems_lock);
+
+ kfree(fcg);
+}
+
+static ssize_t fs_write(struct cgroup *cont, struct cftype *cft,
+  struct file *f, const char __user *ubuf,
+  size_t nbytes, loff_t *pos)
+{
+ int err, add;
+ char buf[64];
+ struct file_system_type *fs;
+ struct file_system_proxy **p, *proxy = NULL;
+ struct fs_cgroup *fcg;
+
+ if (copy_from_user(buf, ubuf, sizeof(buf)))
+  return -EFAULT;
+
+ buf[sizeof(buf) - 1] = '\0';
+ if (buf[0] == '+')
+  add = 1;
+ else if (buf[0] == '-')
+  add = 0;
+ else
+  return -EINVAL;
+
+ if (add) {
+  err = -ENOMEM;
+  proxy = kmalloc(sizeof(*proxy) + sizeof(buf), GFP_KERNEL);
+  if (proxy == NULL)
+   goto err_alloc;
+ }
+
+ fcg = cgroup_to_fs(cont);
+ write_lock(&file_systems_lock);
+
+ for (p = &fcg->fs; *p != NULL; p = &(*p)->next)
+  if (strcmp((*p)->name, buf + 1) == 0)
+   goto fs_exists;
+
+ err = -ESRCH;
+ if (!add)
```

```
+  goto err_unlock;
+
+ for (fs = file_systems; fs != NULL; fs = fs->next)
+  if (strcmp(fs->name, buf + 1) == 0)
+   break;
+
+ if (fs == NULL)
+  goto err_srch;
+
+ if (!try_module_get(fs->owner))
+  goto err_srch;
+
+ proxy->name = (char *)(proxy + 1);
+ strcpy(proxy->name, fs->name);
+ proxy->fs_flags = fs->fs_flags | FS_IS_PROXY;
+ proxy->fs = fs;
+ proxy->next = fcg->fs;
+ fcg->fs = proxy;
+ write_unlock(&file_systems_lock);
+
+ return nbytes;
+
+fs_exists:
+ if (add) {
+  err = -EEXIST;
+  goto err_srch;
+ }
+
+ proxy = *p;
+ *p = proxy->next;
+ write_unlock(&file_systems_lock);
+ put_filesystem(proxy->fs);
+ kfree(proxy);
+ return nbytes;
+
+err_srch:
+ kfree(proxy);
+err_unlock:
+ write_unlock(&file_systems_lock);
+err_alloc:
+ return err;
+}
+
+static ssize_t fs_read(struct cgroup *cont, struct cftype *cft,
+  struct file *f, char __user *ubuf,
+  size_t nbytes, loff_t *pos)
+{
+ int len;
```

```
+ char *buf;
+ struct fs_cgroup *fcg;
+ struct file_system_proxy *p;
+
+ buf = (char *)__get_free_page(GFP_KERNEL);
+
+ len = 0;
+ fcg = cgroup_to_fs(cont);
+ read_lock(&file_systems_lock);
+ for (p = fcg->fs; p != NULL; p = p->next) {
+  if (len + strlen(p->name) > PAGE_SIZE)
+   break;
+
+  len += sprintf(buf + len, "%s\n", p->name);
+ }
+ read_unlock(&file_systems_lock);
+
+ return simple_read_from_buffer(ubuf, nbytes, pos, buf, len);
+}
+
+static struct cftype fs_files[] = {
+ {
+  .name = "list",
+  .write = fs_write,
+  .read = fs_read,
+ },
+};
+
+static int fs_populate(struct cgroup_subsys *ss, struct cgroup *cont)
+{
+ return cgroup_add_files(cont, ss,
+   fs_files, ARRAY_SIZE(fs_files));
+}
+
+struct cgroup_subsys fs_subsys = {
+ .name = "filesystems",
+ .subsys_id = fs_subsys_id,
+ .create = fs_create,
+ .destroy = fs_destroy,
+ .populate = fs_populate,
+};
+#else
+static inline struct file_system_type *task_fs_types(struct task_struct *t)
+{
+ return file_systems;
+}
+#endif
+
```

```
 /* WARNING: This can be used only if we _already_ own a reference */
 void get_filesystem(struct file_system_type *fs)
 {
@@ -41,13 +246,29 @@ void put_filesystem(struct file_system_type *fs)
  module_put(fs->owner);
 }

-static struct file_system_type **find_filesystem(const char *name, unsigned len)
+static struct file_system_type **find_filesystemp(const char *name, unsigned len)
 {
  struct file_system_type **p;
  for (p=&file_systems; *p; p=&(*p)->next)
   if (strlen((*p)->name) == len &&
       strncmp((*p)->name, name, len) == 0)
    break;
+
+ return p;
+}
+
+static struct file_system_type *find_filesystem(const char *name, unsigned len)
+{
+ struct file_system_type *p;
+
+ for (p = task_fs_types(current); p != NULL; p = p->next)
+  if (strlen(p->name) == len &&
+      strncmp(p->name, name, len) == 0)
+   break;
+
+ if (p && (p->fs_flags & FS_IS_PROXY))
+  p = ((struct file_system_proxy *)p)->fs;
+
  return p;
 }

@@ -74,7 +295,7 @@ int register_filesystem(struct file_system_type * fs)
   return -EBUSY;
  INIT_LIST_HEAD(&fs->fs_supers);
  write_lock(&file_systems_lock);
- p = find_filesystem(fs->name, strlen(fs->name));
+ p = find_filesystemp(fs->name, strlen(fs->name));
  if (*p)
   res = -EBUSY;
  else
@@ -131,7 +352,8 @@ static int fs_index(const char __user * __name)

  err = -EINVAL;
  read_lock(&file_systems_lock);
- for (tmp=file_systems, index=0 ; tmp ; tmp=tmp->next, index++) {
```

```
+ for (tmp = task_fs_types(current), index = 0; tmp;
+   tmp = tmp->next, index++) {
   if (strcmp(tmp->name,name) == 0) {
    err = index;
    break;
@@ -148,8 +370,9 @@ static int fs_name(unsigned int index, char __user * buf)
  int len, res;

  read_lock(&file_systems_lock);
- for (tmp = file_systems; tmp; tmp = tmp->next, index--)
-  if (index <= 0 && try_module_get(tmp->owner))
+ for (tmp = task_fs_types(current); tmp; tmp = tmp->next, index--)
+  if (index <= 0 && ((tmp->fs_flags & FS_IS_PROXY) ||
+     try_module_get(tmp->owner)))
    break;
  read_unlock(&file_systems_lock);
  if (!tmp)
@@ -158,7 +381,8 @@ static int fs_name(unsigned int index, char __user * buf)
  /* OK, we got the reference, so we can safely block */
  len = strlen(tmp->name) + 1;
  res = copy_to_user(buf, tmp->name, len) ? -EFAULT : 0;
- put_filesystem(tmp);
+ if (!(tmp->fs_flags & FS_IS_PROXY))
+  put_filesystem(tmp);
  return res;
 }

@@ -168,7 +392,8 @@ static int fs_maxindex(void)
  int index;

  read_lock(&file_systems_lock);
- for (tmp = file_systems, index = 0 ; tmp ; tmp = tmp->next, index++)
+ for (tmp = task_fs_types(current), index = 0; tmp;
+   tmp = tmp->next, index++)
   ;
  read_unlock(&file_systems_lock);
  return index;
@@ -203,7 +428,7 @@ int get_filesystem_list(char * buf)
  struct file_system_type * tmp;

  read_lock(&file_systems_lock);
- tmp = file_systems;
+ tmp = task_fs_types(current);
  while (tmp && len < PAGE_SIZE - 80) {
   len += sprintf(buf+len, "%s\t%s\n",
    (tmp->fs_flags & FS_REQUIRES_DEV) ? "" : "nodev",
@@ -221,13 +446,13 @@ struct file_system_type *get_fs_type(const char *name)
  unsigned len = dot ? dot - name : strlen(name);
```

```
  read_lock(&file_systems_lock);
- fs = *(find_filesystem(name, len));
+ fs = find_filesystem(name, len);
  if (fs && !try_module_get(fs->owner))
   fs = NULL;
  read_unlock(&file_systems_lock);
  if (!fs && (request_module("%.*s", len, name) == 0)) {
   read_lock(&file_systems_lock);
-  fs = *(find_filesystem(name, len));
+  fs = find_filesystem(name, len);
   if (fs && !try_module_get(fs->owner))
    fs = NULL;
   read_unlock(&file_systems_lock);
diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
index 228235c..cda4be7 100644
--- a/include/linux/cgroup_subsys.h
+++ b/include/linux/cgroup_subsys.h
@@ -42,3 +42,9 @@ SUBSYS(mem_cgroup)
 #endif

 /* */
+
+#ifdef CONFIG_CGROUP_FS
+SUBSYS(fs)
+#endif
+
+/* */
diff --git a/include/linux/fs.h b/include/linux/fs.h
index a456afe..19b6e0e 100644
--- a/include/linux/fs.h
+++ b/include/linux/fs.h
@@ -93,6 +93,7 @@ extern int dir_notify_enable;
 #define FS_REQUIRES_DEV 1
 #define FS_BINARY_MOUNTDATA 2
 #define FS_HAS_SUBTYPE 4
+#define FS_IS_PROXY 4096 /* This is a proxy for control groups */
 #define FS_REVAL_DOT 16384 /* Check the paths ".", ".." for staleness */
 #define FS_RENAME_DOES_D_MOVE 32768 /* FS will handle d_move()
       * during rename() internally.
@@ -1426,12 +1427,12 @@ int sync_inode(struct inode *inode, struct writeback_control *wbc);

 struct file_system_type {
  const char *name;
+ struct file_system_type * next;
  int fs_flags;
  int (*get_sb) (struct file_system_type *, int,
         const char *, void *, struct vfsmount *);
```

```
  void (*kill_sb) (struct super_block *);
  struct module *owner;
- struct file_system_type * next;
  struct list_head fs_supers;

  struct lock_class_key s_lock_key;
diff --git a/init/Kconfig b/init/Kconfig
index 732a1c2..60d5f08 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -292,6 +292,12 @@ config CGROUP_NS
          for instance virtual servers and checkpoint/restart
          jobs.

+config CGROUP_FS
+ bool "Filesystems control group"
+ depends on CGROUPS
+ help
+   Tune the filesystems visibility
+
 config CPUSETS
  bool "Cpuset support"
  depends on SMP && CGROUPS
```

_____

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