Subject: [PATCH 9/13] Make proc be able to have multiple super blocks
Posted by Pavel Emelianov on Thu, 24 May 2007 13:01:24 GMT
View Forum Message <> Reply to Message

Each namespace must have its own proc super block where
dentries and inodes representing the tasks live.

Plus proc super block bust hold the namespace it draws
the pids from.

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

```
diff --git a/include/linux/proc_fs.h b/include/linux/proc_fs.h
index 38f7082..d107a33 100644
--- a/include/linux/proc_fs.h
+++ b/include/linux/proc_fs.h
@@ -126,7 +126,8 @@ extern struct proc_dir_entry *create_pro
 extern void remove_proc_entry(const char *name, struct proc_dir_entry *parent);

 extern struct vfsmount *proc_mnt;
-extern int proc_fill_super(struct super_block *,void *,int);
+struct pid_namespace;
+extern int proc_fill_super(struct super_block *, struct pid_namespace *);
 extern struct inode *proc_get_inode(struct super_block *, unsigned int, struct proc_dir_entry *);

 /*
diff --git a/fs/proc/inode.c b/fs/proc/inode.c
index 2ba47e4..5a7b5d5 100644
--- a/fs/proc/inode.c
+++ b/fs/proc/inode.c
@@ -15,6 +15,7 @@
 #include <linux/init.h>
 #include <linux/module.h>
 #include <linux/smp_lock.h>
+#include <linux/pid_namespace.h>

 #include <asm/system.h>
 #include <asm/uaccess.h>
@@ -428,9 +429,17 @@ out_mod:
  return NULL;
 }

-int proc_fill_super(struct super_block *s, void *data, int silent)
+int proc_fill_super(struct super_block *s, struct pid_namespace *ns)
 {
  struct inode * root_inode;
```

```
+ struct proc_dir_entry * root_dentry;
+
+ root_dentry = &proc_root;
+ if (ns != &init_pid_ns) {
+  root_dentry = create_proc_root();
+  if (root_dentry == NULL)
+   goto out_no_de;
+ }

  s->s_flags |= MS_NODIRATIME | MS_NOSUID | MS_NOEXEC;
  s->s_blocksize = 1024;
@@ -439,8 +448,8 @@ int proc_fill_super(struct super_block *
  s->s_op = &proc_sops;
  s->s_time_gran = 1;

- de_get(&proc_root);
- root_inode = proc_get_inode(s, PROC_ROOT_INO, &proc_root);
+ de_get(root_dentry);
+ root_inode = proc_get_inode(s, PROC_ROOT_INO, root_dentry);
  if (!root_inode)
   goto out_no_root;
  root_inode->i_uid = 0;
@@ -451,9 +460,10 @@ int proc_fill_super(struct super_block *
  return 0;

 out_no_root:
- printk("proc_read_super: get root inode failed\n");
  iput(root_inode);
- de_put(&proc_root);
+ de_put(root_dentry);
+out_no_de:
+ printk("proc_read_super: get root inode failed\n");
  return -ENOMEM;
 }
 MODULE_LICENSE("GPL");
diff --git a/fs/proc/internal.h b/fs/proc/internal.h
index 10f3601..b981956 100644
--- a/fs/proc/internal.h
+++ b/fs/proc/internal.h
@@ -71,3 +71,5 @@ static inline int proc_fd(struct inode *
 {
  return PROC_I(inode)->fd;
 }
+
+struct proc_dir_entry * create_proc_root(void);
diff --git a/fs/proc/root.c b/fs/proc/root.c
index 41f1703..e697f45 100644
--- a/fs/proc/root.c
```

```
+++ b/fs/proc/root.c
@@ -18,32 +18,76 @@
 #include <linux/bitops.h>
 #include <linux/smp_lock.h>
 #include <linux/mount.h>
+#include <linux/pid_namespace.h>

 #include "internal.h"

 struct proc_dir_entry *proc_net, *proc_net_stat, *proc_bus, *proc_root_fs, *proc_root_driver;

+static int proc_test_super(struct super_block *sb, void *data)
+{
+ return sb->s_fs_info == data;
+}
+
+static int proc_set_super(struct super_block *sb, void *data)
+{
+ get_pid_ns((struct pid_namespace *)data);
+ sb->s_fs_info = data;
+ return set_anon_super(sb, NULL);
+}
+
 static int proc_get_sb(struct file_system_type *fs_type,
  int flags, const char *dev_name, void *data, struct vfsmount *mnt)
 {
+ int err;
+ struct super_block *sb;
+ struct pid_namespace *ns;
+ struct proc_inode *ei;
+
  if (proc_mnt) {
   /* Seed the root directory with a pid so it doesn't need
    * to be special in base.c.  I would do this earlier but
    * the only task alive when /proc is mounted the first time
    * is the init_task and it doesn't have any pids.
    */
-  struct proc_inode *ei;
   ei = PROC_I(proc_mnt->mnt_sb->s_root->d_inode);
   if (!ei->pid)
    ei->pid = find_get_pid(1);
  }
- return get_sb_single(fs_type, flags, data, proc_fill_super, mnt);
+
+ ns = current->nsproxy->pid_ns;
+ sb = sget(fs_type, proc_test_super, proc_set_super, ns);
+ if (IS_ERR(sb))
+  return PTR_ERR(sb);
```

```
+
+ if (!sb->s_root) {
+  sb->s_flags = flags;
+  err = proc_fill_super(sb, ns);
+  if (err) {
+   up_write(&sb->s_umount);
+   deactivate_super(sb);
+   return err;
+  }
+
+  ei = PROC_I(sb->s_root->d_inode);
+  if (!ei->pid)
+   ei->pid = find_get_pid(1);
+  sb->s_flags |= MS_ACTIVE;
+ }
+
+ return simple_set_mnt(mnt, sb);
+}
+
+static void proc_kill_sb(struct super_block *sb)
+{
+ put_pid_ns((struct pid_namespace *)sb->s_fs_info);
+ kill_anon_super(sb);
 }

 static struct file_system_type proc_fs_type = {
  .name  = "proc",
  .get_sb  = proc_get_sb,
- .kill_sb = kill_anon_super,
+ .kill_sb = proc_kill_sb,
 };

 void __init proc_root_init(void)
@@ -153,6 +197,24 @@ struct proc_dir_entry proc_root = {
  .parent  = &proc_root,
 };

+struct proc_dir_entry * create_proc_root(void)
+{
+ struct proc_dir_entry *de;
+
+ de = kzalloc(sizeof(struct proc_dir_entry), GFP_KERNEL);
+ if (de != NULL) {
+  de->low_ino = PROC_ROOT_INO;
+  de->namelen = 5;
+  de->name = "/proc";
+  de->mode = S_IFDIR | S_IRUGO | S_IXUGO;
+  de->nlink = 2;
```

```
+  de->proc_iops = &proc_root_inode_operations;
+  de->proc_fops = &proc_root_operations;
+  de->parent = de;
+ }
+ return de;
+}
+
 EXPORT_SYMBOL(proc_symlink);
 EXPORT_SYMBOL(proc_mkdir);
 EXPORT_SYMBOL(create_proc_entry);
```