Subject: [PATCH 5/16] Make proc be mountable from different pid namespaces
Posted by Pavel Emelianov on Fri, 06 Jul 2007 08:05:42 GMT
View Forum Message <> Reply to Message

Each pid namespace should have the proc_mnt pointer even when
there's no user mounts to make proc_flush_task() work. To do
this we call the kern_mount() to obtain the proc mount point.

Since the current pid_namespace during this call is not the
newly created one we use the introduced MS_KERNMOUNT flag
to pass the namespace pointer to the proc_get_sb() call.

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

```
 fs/proc/inode.c             |   20 +++++--
 fs/proc/internal.h          |    2
 fs/proc/root.c              |  116 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++--
 include/linux/pid_namespace.h |    3 +
 include/linux/proc_fs.h     |   15 +++++
 5 files changed, 147 insertions(+), 9 deletions(-)

diff -upr linux-2.6.22-rc4-mm2.orig/fs/proc/inode.c linux-2.6.22-rc4-mm2-2/fs/proc/inode.c
--- linux-2.6.22-rc4-mm2.orig/fs/proc/inode.c 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/fs/proc/inode.c 2007-07-04 19:00:38.000000000 +0400
@@ -16,6 +16,7 @@
 #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>
@@ -429,9 +430,17 @@ 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;
@@ -440,8 +449,8 @@ 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;
@@ -452,9 +461,10 @@ 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 -upr linux-2.6.22-rc4-mm2.orig/fs/proc/internal.h linux-2.6.22-rc4-mm2-2/fs/proc/internal.h
--- linux-2.6.22-rc4-mm2.orig/fs/proc/internal.h 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/fs/proc/internal.h 2007-07-04 19:00:38.000000000 +0400
@@ -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 -upr linux-2.6.22-rc4-mm2.orig/fs/proc/root.c linux-2.6.22-rc4-mm2-2/fs/proc/root.c
--- linux-2.6.22-rc4-mm2.orig/fs/proc/root.c 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/fs/proc/root.c 2007-07-04 19:00:39.000000000 +0400
@@ -18,32 +18,89 @@
 #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)
+{
+ struct pid_namespace *ns;
+
+ ns = (struct pid_namespace *)data;
+ sb->s_fs_info = get_pid_ns(ns);
+ 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);
+
+ if (flags & MS_KERNMOUNT)
+  ns = (struct pid_namespace *)data;
+ else
+  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;
+
+  mntput(ns->proc_mnt);
+  ns->proc_mnt = mnt;
+ }
+
+ return simple_set_mnt(mnt, sb);
+}
+
+static void proc_kill_sb(struct super_block *sb)
+{
+ struct pid_namespace *ns;
+
+ ns = (struct pid_namespace *)sb->s_fs_info;
+ kill_anon_super(sb);
+ if (ns != NULL)
+  put_pid_ns(ns);
 }

 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)
@@ -60,6 +117,7 @@ void __init proc_root_init(void)
   unregister_filesystem(&proc_fs_type);
   return;
  }
+
  proc_misc_init();
  proc_net = proc_mkdir("net", NULL);
  proc_net_stat = proc_mkdir("net/stat", NULL);
@@ -153,6 +211,58 @@ struct proc_dir_entry proc_root = {
  .parent  = &proc_root,
```

```
 };

+/*
+ * creates the proc root entry for different proc trees
+ */
+
+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;
+}
+
+int pid_ns_prepare_proc(struct pid_namespace *ns)
+{
+ struct vfsmount *mnt;
+
+ mnt = kern_mount_data(&proc_fs_type, ns);
+ if (!IS_ERR(mnt))
+  /*
+   * do not save the reference from the proc super
+   * block to the namespace. otherwise we will get
+   * a circular reference ns->proc_mnt->mnt_sb->ns
+   */
+  put_pid_ns(ns);
+ return 0;
+}
+
+void pid_ns_release_proc(struct pid_namespace *ns)
+{
+ struct vfsmount *mnt;
+
+ mnt = ns->proc_mnt;
+ /*
+  * do not put the namespace reference as it wa not get in
+  * pid_ns_prepare_proc(). safe to set NULL here as this
+  * namespace is already dead and all the proc mounts are
```

```
+  * released so nobudo will se this super block
+  */
+ mnt->mnt_sb->s_fs_info = NULL;
+ mntput(mnt);
+}
+
 EXPORT_SYMBOL(proc_symlink);
 EXPORT_SYMBOL(proc_mkdir);
 EXPORT_SYMBOL(create_proc_entry);
diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid_namespace.h
linux-2.6.22-rc4-mm2-2/include/linux/pid_namespace.h
--- linux-2.6.22-rc4-mm2.orig/include/linux/pid_namespace.h 2007-06-14 12:14:29.000000000
+0400
+++ linux-2.6.22-rc4-mm2-2/include/linux/pid_namespace.h 2007-07-04 19:00:39.000000000
+0400
@@ -16,6 +15,9 @@ struct pidmap {
  struct task_struct *child_reaper;
  struct kmem_cache *pid_cachep;
  struct pid_namespace *parent;
+#ifdef CONFIG_PROC_FS
+ struct vfsmount *proc_mnt;
+#endif
 };

 extern struct pid_namespace init_pid_ns;
diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/proc_fs.h
linux-2.6.22-rc4-mm2-2/include/linux/proc_fs.h
--- linux-2.6.22-rc4-mm2.orig/include/linux/proc_fs.h 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/include/linux/proc_fs.h 2007-07-04 19:00:38.000000000 +0400
@@ -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 *);

 /*
@@ -143,6 +144,9 @@ extern const struct file_operations proc
 extern const struct file_operations proc_kmsg_operations;
 extern const struct file_operations ppc_htab_operations;

+extern int pid_ns_prepare_proc(struct pid_namespace *ns);
+extern void pid_ns_release_proc(struct pid_namespace *ns);
+
 /*
  * proc_tty.c
```

```
 */
@@ -248,6 +254,15 @@ static inline void proc_tty_unregister_d

 extern struct proc_dir_entry proc_root;

+static inline int pid_ns_prepare_proc(struct pid_namespace *ns)
+{
+ return 0;
+}
+
+static inline void pid_ns_release_proc(struct pid_namespace *ns)
+{
+}
+
 #endif /* CONFIG_PROC_FS */

 #if !defined(CONFIG_PROC_KCORE)
```

_____

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