Subject: [PATCH 15/20] Initialize the namespace's proc_mnt
Posted by Pavel Emelianov on Fri, 10 Aug 2007 11:48:20 GMT
View Forum Message <> Reply to Message

The namespace's proc_mnt must be kern_mount-ed to make this
pointer always valid, independently of whether the user space
mounted the proc or not. This solves raced in proc_flush_task,
etc. with the proc_mnt switching from NULL to not-NULL.

The initialization is done after the init's pid is created and
hashed to make proc_get_sb() finr it and get for root inode.

Sice the namespace holds the vfsmnt, vfsmnt holds the superblock
and the superblock holds the namespace we must explicitly break
this circle to destroy all the stuff. This is done after the init
of the namespace dies. Running a few steps forward - when init
exits it will kill all its children, so no proc_mnt will be
needed after its death.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
Cs: Oleg Nesterov <oleg@tv-sign.ru>

```
 fs/proc/base.c          |    4 ++++
 fs/proc/root.c          |   16 +++++++++++++++++
 include/linux/proc_fs.h |   12 ++++++++++++
 kernel/fork.c           |    7 +++++++
 4 files changed, 39 insertions(+)

--- ./fs/proc/base.c.ve12 2007-08-06 14:58:51.000000000 +0400
+++ ./fs/proc/base.c 2007-08-06 14:58:51.000000000 +0400
@@ -2170,6 +2170,10 @@ out:
   proc_flush_task_mnt(upid->ns->proc_mnt, upid->nr,
     leader ? 0 : tgid->numbers[i].nr);
  }
+
+ upid = &pid->numbers[pid->level];
+ if (upid->nr == 1)
+  pid_ns_release_proc(upid->ns);
 }

 static struct dentry *proc_pid_instantiate(struct inode *dir,
--- ./fs/proc/root.c.ve12 2007-08-06 14:54:42.000000000 +0400
+++ ./fs/proc/root.c 2007-08-06 14:58:51.000000000 +0400
@@ -153,6 +212,22 @@ struct proc_dir_entry proc_root = {
  .parent  = &proc_root,
 };
```

```
+int pid_ns_prepare_proc(struct pid_namespace *ns)
+{
+ struct vfsmount *mnt;
+
+ mnt = kern_mount_data(&proc_fs_type, ns);
+ if (IS_ERR(mnt))
+  return PTR_ERR(mnt);
+
+ return 0;
+}
+
+void pid_ns_release_proc(struct pid_namespace *ns)
+{
+ mntput(ns->proc_mnt);
+}
+
 EXPORT_SYMBOL(proc_symlink);
 EXPORT_SYMBOL(proc_mkdir);
 EXPORT_SYMBOL(create_proc_entry);
--- ./include/linux/proc_fs.h.ve12 2007-08-06 14:58:51.000000000 +0400
+++ ./include/linux/proc_fs.h 2007-08-06 14:58:51.000000000 +0400
@@ -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
  */
@@ -250,6 +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)
```

```
--- ./kernel/fork.c.ve12 2007-08-06 14:58:51.000000000 +0400
+++ ./kernel/fork.c 2007-08-06 14:58:51.000000000 +0400
@@ -50,6 +50,7 @@
 #include <linux/taskstats_kern.h>
 #include <linux/random.h>
 #include <linux/tty.h>
+#include <linux/proc_fs.h>

 #include <asm/pgtable.h>
 #include <asm/pgalloc.h>
@@ -1134,6 +1135,12 @@ static struct task_struct *copy_process(
   pid = alloc_pid(task_active_pid_ns(p));
   if (!pid)
    goto bad_fork_cleanup_namespaces;
+
+  if (clone_flags & CLONE_NEWPID) {
+   retval = pid_ns_prepare_proc(task_active_pid_ns(p));
+   if (retval < 0)
+    goto bad_fork_free_pid;
+  }
  }

  p->pid = pid_nr(pid);
```