Subject: [PATCH 6/12] pid: Make session_of_pgrp use struct pid instead of pid_t.
Posted by ebiederm on Wed, 13 Dec 2006 11:07:50 GMT
View Forum Message <> Reply to Message

To properly implement a pid namespace I need to deal exclusively
in terms of struct pid, because pid_t values become ambiguous.

To this end session_of_pgrp is transformed to take and return
a struct pid pointer. To avoid the need to worry about reference
counting I now require my caller to hold the appropriate locks.
Leaving callers repsonsible for increasing the reference count
if they need access to the result outside of the locks.

Since session_of_pgrp currently only has one caller and that
caller simply uses only test the result for equality with
another process group, the locking change means I don't
actually have to acquire the tasklist_lock at all.

tiocspgrp is also modified to take and release the lock. The logic
there is a little more complicated but nothing I won't need when I
convert pgrp of a tty to a struct pid pointer.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
drivers/char/tty_io.c | 24 ++++++++++++++++++-------
include/linux/kernel.h | 3 ++-
kernel/exit.c | 16 +++++++---------
3 files changed, 26 insertions(+), 17 deletions(-)

diff --git a/drivers/char/tty_io.c b/drivers/char/tty_io.c
index 628925e..74f078a 100644
--- a/drivers/char/tty_io.c
+++ b/drivers/char/tty_io.c
@@ -2990,7 +2990,8 @@ static int tiocgpgrp(struct tty_struct *tty, struct tty_struct *real_tty, pid_t

static int tiocspgrp(struct tty_struct *tty, struct tty_struct *real_tty, pid_t __user *p)
{
- pid_t pgrp;
+ struct pid *pgrp;
+ pid_t pgrp_nr;
int retval = tty_check_change(real_tty);

if (retval == -EIO)
@@ -3001,14 +3002,23 @@ static int tiocspgrp(struct tty_struct *tty, struct tty_struct *real_tty,
pid_t
(current->signal->tty != real_tty) ||
(real_tty->session != process_session(current)))
return -ENOTTY;

```
- if (get_user(pgrp, p))
+ if (get_user(pgrp_nr, p))
   return -EFAULT;
- if (pgrp < 0)
+ if (pgrp_nr < 0)
   return -EINVAL;
- if (session_of_pgrp(pgrp) != process_session(current))
-  return -EPERM;
- real_tty->pgrp = pgrp;
- return 0;
+ rcu_read_lock();
+ pgrp = find_pid(pgrp_nr);
+ retval = -ESRCH;
+ if (!pgrp)
+  goto out_unlock;
+ retval = -EPERM;
+ if (session_of_pgrp(pgrp) != task_session(current))
+  goto out_unlock;
+ retval = 0;
+ real_tty->pgrp = pgrp_nr;
+out_unlock:
+ rcu_read_unlock();
+ return retval;
 }

 /**
diff --git a/include/linux/kernel.h b/include/linux/kernel.h
index e8bfac3..ac52d33 100644
--- a/include/linux/kernel.h
+++ b/include/linux/kernel.h
@@ -138,7 +138,8 @@ extern unsigned long long memparse(char *ptr, char **retptr);
 extern int core_kernel_text(unsigned long addr);
 extern int __kernel_text_address(unsigned long addr);
 extern int kernel_text_address(unsigned long addr);
-extern int session_of_pgrp(int pgrp);
+struct pid;
+extern struct pid *session_of_pgrp(struct pid *pgrp);

 extern void dump_thread(struct pt_regs *regs, struct user *dump);

diff --git a/kernel/exit.c b/kernel/exit.c
index 122fadb..97117e7 100644
--- a/kernel/exit.c
+++ b/kernel/exit.c
@@ -185,21 +185,19 @@ repeat:
  * This checks not only the pgrp, but falls back on the pid if no
  * satisfactory pgrp is found. I dunno - gdb doesn't work correctly
  * without this...
```

```
+ *
+ * The caller must hold rcu lock or the tasklist lock.
  */
-int session_of_pgrp(int pgrp)
+struct pid *session_of_pgrp(struct pid *pgrp)
 {
  struct task_struct *p;
- int sid = 0;
-
- read_lock(&tasklist_lock);
+ struct pid *sid = NULL;

- p = find_task_by_pid_type(PIDTYPE_PGID, pgrp);
+ p = pid_task(pgrp, PIDTYPE_PGID);
  if (p == NULL)
-  p = find_task_by_pid(pgrp);
+  p = pid_task(pgrp, PIDTYPE_PID);
  if (p != NULL)
-  sid = process_session(p);
-
- read_unlock(&tasklist_lock);
+  sid = task_session(p);

  return sid;
 }
--
1.4.4.1.g278f
```

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers