Subject: [PATCH 0/12] tty layer and misc struct pid conversions
Posted by ebiederm on Wed, 13 Dec 2006 11:03:39 GMT
View Forum Message <> Reply to Message

The aim of this patch set is to start wrapping up the struct pid
conversions. As such this patchset culminates with the removal
of kill_pg, kill_pg_info, __kill_pg_info, do_each_task_pid, and
while_each_task_pid.

kill_proc, daemonize, and kernel_thread are still in my sights but
there is still work to get to them.

The first three are basic cleanups around disassociate_ctty,
while working on converting it I found several issues. tty_old_pgrp
can be a tricky concept to wrap your head around.

1 tty: Make __proc_set_tty static.
2 tty: Clarify disassociate_ctty
3 tty: Fix the locking for signal->session in disassociate_ctty

These just stop using the old helper functions.

4 signal: Use kill_pgrp not kill_pg in the sunos compatibility code.
5 signal: Rewrite kill_something_info so it uses newer helpers.

Then the grind to convert the tty layer and all of it's helper
functions to struct pid.

6 pid: Make session_of_pgrp use struct pid instead of pid_t.
7 pid: Use struct pid for talking about process groups in exit.c
8 pid: Replace is_orphaned_pgrp with is_current_pgrp_orphaned
9 tty: Update the tty layer to work with struct pid.

A final helper function update.

10 pid: Replace do/while_each_task_pid with do/while_each_pid_task

And the removal of the functions that are now unused.
11 pid: Remove now unused do_each_task_pid and while_each_task_pid
12 pid: Remove the now unused kill_pg kill_pg_info and __kill_pg_info

All of these should be fairly simple and to the point.

Eric

_____
Containers mailing list
Containers@lists.osdl.org

Subject: [PATCH 1/12] tty: Make __proc_set_tty static.
Posted by ebiederm on Wed, 13 Dec 2006 11:07:45 GMT
View Forum Message <> Reply to Message

Currently all users of __proc_set_tty are in tty_io.c so
make the function static.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
drivers/char/tty_io.c | 3 ++-
include/linux/tty.h | 1 -
2 files changed, 2 insertions(+), 2 deletions(-)

```
diff --git a/drivers/char/tty_io.c b/drivers/char/tty_io.c
index a333881..bb09e4a 100644
--- a/drivers/char/tty_io.c
+++ b/drivers/char/tty_io.c
@@ -155,6 +155,7 @@ int tty_ioctl(struct inode * inode, struct file * file,
       unsigned int cmd, unsigned long arg);
 static int tty_fasync(int fd, struct file * filp, int on);
 static void release_mem(struct tty_struct *tty, int idx);
+static void __proc_set_tty(struct task_struct *tsk, struct tty_struct *tty);

 /**
  * alloc_tty_struct - allocate a tty object
@@ -3796,7 +3797,7 @@ void proc_clear_tty(struct task_struct *p)
 }
 EXPORT_SYMBOL(proc_clear_tty);

-void __proc_set_tty(struct task_struct *tsk, struct tty_struct *tty)
+static void __proc_set_tty(struct task_struct *tsk, struct tty_struct *tty)
 {
  if (tty) {
   tty->session = process_session(tsk);
diff --git a/include/linux/tty.h b/include/linux/tty.h
index 0161a8c..1185bca 100644
--- a/include/linux/tty.h
+++ b/include/linux/tty.h
@@ -313,7 +313,6 @@ extern int tty_ioctl(struct inode *inode, struct file *file, unsigned int cmd,

 extern dev_t tty_devnum(struct tty_struct *tty);
 extern void proc_clear_tty(struct task_struct *p);
-extern void __proc_set_tty(struct task_struct *tsk, struct tty_struct *tty);
 extern void proc_set_tty(struct task_struct *tsk, struct tty_struct *tty);
 extern struct tty_struct *get_current_tty(void);
```

--
1.4.4.1.g278f

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

Subject: [PATCH 2/12] tty: Clarify disassociate_ctty
Posted by ebiederm on Wed, 13 Dec 2006 11:07:46 GMT
View Forum Message <> Reply to Message

The code to look at tty_old_pgrp and send SIGHUP and SIGCONT
when it is present only executes when disassociate_ctty is
called from do_exit. Make this clear by adding an explict
on_exit check, and explicitly setting tty_old_pgrp to 0.

In addition fix the locking by reading tty_old_pgrp under
the siglock.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/char/tty_io.c | 8 ++++++--
 1 files changed, 6 insertions(+), 2 deletions(-)

diff --git a/drivers/char/tty_io.c b/drivers/char/tty_io.c
index bb09e4a..acb2f5d 100644
--- a/drivers/char/tty_io.c
+++ b/drivers/char/tty_io.c
@@ -1508,8 +1508,12 @@ void disassociate_ctty(int on_exit)
   /* XXX: here we race, there is nothing protecting tty */
   if (on_exit && tty->driver->type != TTY_DRIVER_TYPE_PTY)
    tty_vhangup(tty);
- } else {
- pid_t old_pgrp = current->signal->tty_old_pgrp;
+ } else if (on_exit) {
+ pid_t old_pgrp;
+ spin_lock_irq(&current->sighand->siglock);
+ old_pgrp = current->signal->tty_old_pgrp;
+ current->signal->tty_old_pgrp = 0;
+ spin_unlock_irq(&current->sighand->siglock);
   if (old_pgrp) {
    kill_pg(old_pgrp, SIGHUP, on_exit);
    kill_pg(old_pgrp, SIGCONT, on_exit);
--
1.4.4.1.g278f

Subject: [PATCH 3/12] tty: Fix the locking for signal->session in disassociate_ctty
Posted by ebiederm on Wed, 13 Dec 2006 11:07:47 GMT
View Forum Message <> Reply to Message

commit 24ec839c431eb79bb8f6abc00c4e1eb3b8c4d517 while fixing
the locking for signal->tty got the locking wrong for
signal->session. This places our accesses of signal->session
back under the tasklist_lock where they belong.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
drivers/char/tty_io.c | 4 +---
1 files changed, 1 insertions(+), 3 deletions(-)

```
diff --git a/drivers/char/tty_io.c b/drivers/char/tty_io.c
index acb2f5d..628925e 100644
--- a/drivers/char/tty_io.c
+++ b/drivers/char/tty_io.c
@@ -1496,7 +1496,6 @@ void disassociate_ctty(int on_exit)
 {
  struct tty_struct *tty;
  int tty_pgrp = -1;
- int session;

  lock_kernel();

@@ -1530,7 +1529,6 @@ void disassociate_ctty(int on_exit)

  spin_lock_irq(&current->sighand->siglock);
  current->signal->tty_old_pgrp = 0;
- session = process_session(current);
  spin_unlock_irq(&current->sighand->siglock);

  mutex_lock(&tty_mutex);
@@ -1549,7 +1547,7 @@ void disassociate_ctty(int on_exit)

  /* Now clear signal->tty under the lock */
  read_lock(&tasklist_lock);
- session_clear_tty(session);
+ session_clear_tty(process_session(current));
  read_unlock(&tasklist_lock);
```

```
 unlock_kernel();
 }
--
1.4.4.1.g278f
```

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

Subject: [PATCH 4/12] signal: Use kill_pgrp not kill_pg in the sunos compatibility code.
Posted by ebiederm on Wed, 13 Dec 2006 11:07:48 GMT
View Forum Message <> Reply to Message

I am slowly moving to a model where all process killing
is struct pid based instead of pid_t based. The sunos
compatibility code is one of the last users of the old pid_t
based kill_pg in the kernel. By being complete I allow
for the future removal of kill_pg from the kernel, which
will ensure I don't miss something.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 arch/sparc/kernel/sys_sunos.c | 10 ++++++----
 arch/sparc64/kernel/sys_sunos32.c | 11 ++++++++++--
 2 files changed, 15 insertions(+), 6 deletions(-)

```
diff --git a/arch/sparc/kernel/sys_sunos.c b/arch/sparc/kernel/sys_sunos.c
index 0bf8c16..da6606f 100644
--- a/arch/sparc/kernel/sys_sunos.c
+++ b/arch/sparc/kernel/sys_sunos.c
@@ -859,14 +859,16 @@ asmlinkage int sunos_wait4(pid_t pid, unsigned int __user *stat_addr,
  return ret;
 }

-extern int kill_pg(int, int, int);
 asmlinkage int sunos_killpg(int pgrp, int sig)
 {
  int ret;

- lock_kernel();
- ret = kill_pg(pgrp, sig, 0);
- unlock_kernel();
+ rcu_read_lock();
+ ret = -EINVAL;
+ if (pgrp > 0)
```

```
+  ret = kill_pgrp(find_pid(pgrp), sig, 0);
+ rcu_read_unlock();
+
  return ret;
 }

diff --git a/arch/sparc64/kernel/sys_sunos32.c b/arch/sparc64/kernel/sys_sunos32.c
index 4446f66..d9280b1 100644
--- a/arch/sparc64/kernel/sys_sunos32.c
+++ b/arch/sparc64/kernel/sys_sunos32.c
@@ -824,10 +824,17 @@ asmlinkage int sunos_wait4(compat_pid_t pid, compat_uint_t __user
*stat_addr, in
  return ret;
 }

-extern int kill_pg(int, int, int);
 asmlinkage int sunos_killpg(int pgrp, int sig)
 {
- return kill_pg(pgrp, sig, 0);
+ int ret;
+
+ rcu_read_lock();
+ ret = -EINVAL;
+ if (pgrp > 0)
+  ret = kill_pgrp(find_pid(pgrp), sig, 0);
+ rcu_read_unlock();
+
+ return ret;
 }

 asmlinkage int sunos_audit(void)
--
1.4.4.1.g278f
```

Subject: [PATCH 5/12] signal: Rewrite kill_something_info so it uses newer helpers.
Posted by ebiederm on Wed, 13 Dec 2006 11:07:49 GMT
View Forum Message <> Reply to Message

The goal is to remove users of the old signal helper functions
so they can be removed.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
---
 kernel/signal.c |   12 ++++++++----
 1 files changed, 8 insertions(+), 4 deletions(-)

diff --git a/kernel/signal.c b/kernel/signal.c
index 1921ffd..1e34d32 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -1192,8 +1192,10 @@ EXPORT_SYMBOL_GPL(kill_pid_info_as_uid);

 static int kill_something_info(int sig, struct siginfo *info, int pid)
 {
+ int ret;
+ rcu_read_lock();
  if (!pid) {
-  return kill_pg_info(sig, info, process_group(current));
+  ret = kill_pgrp_info(sig, info, task_pgrp(current));
  } else if (pid == -1) {
   int retval = 0, count = 0;
   struct task_struct * p;
@@ -1208,12 +1210,14 @@ static int kill_something_info(int sig, struct siginfo *info, int pid)
    }
   }
   read_unlock(&tasklist_lock);
-  return count ? retval : -ESRCH;
+  ret = count ? retval : -ESRCH;
  } else if (pid < 0) {
-  return kill_pg_info(sig, info, -pid);
+  ret = kill_pgrp_info(sig, info, find_pid(-pid));
  } else {
-  return kill_proc_info(sig, info, pid);
+  ret = kill_pid_info(sig, info, find_pid(pid));
  }
+ rcu_read_unlock();
+ return ret;
 }

 /*
--
1.4.4.1.g278f
```

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

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
```


Subject: [PATCH 7/12] pid: Use struct pid for talking about process groups in exit.c
Posted by ebiederm on Wed, 13 Dec 2006 11:07:51 GMT
View Forum Message <> Reply to Message

Modify has_stopped_jobs and will_become_orphan_pgrp
to use struct pid based process groups. This reduces
the number of hash tables looks ups and paves the way
for multiple pid spaces.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 kernel/exit.c | 42 ++++++++++++++++++++++++++----------------
 1 files changed, 22 insertions(+), 20 deletions(-)

diff --git a/kernel/exit.c b/kernel/exit.c

```
index 97117e7..5f8455e 100644
--- a/kernel/exit.c
+++ b/kernel/exit.c
@@ -210,22 +210,22 @@ struct pid *session_of_pgrp(struct pid *pgrp)
  *
  * "I ask you, have you ever known what it is to be an orphan?"
  */
-static int will_become_orphaned_pgrp(int pgrp, struct task_struct *ignored_task)
+static int will_become_orphaned_pgrp(struct pid *pgrp, struct task_struct *ignored_task)
 {
  struct task_struct *p;
  int ret = 1;

- do_each_task_pid(pgrp, PIDTYPE_PGID, p) {
+ do_each_pid_task(pgrp, PIDTYPE_PGID, p) {
    if (p == ignored_task
      || p->exit_state
      || is_init(p->real_parent))
     continue;
-   if (process_group(p->real_parent) != pgrp &&
-       process_session(p->real_parent) == process_session(p)) {
+   if (task_pgrp(p->real_parent) != pgrp &&
+       task_session(p->real_parent) == task_session(p)) {
     ret = 0;
     break;
    }
- } while_each_task_pid(pgrp, PIDTYPE_PGID, p);
+ } while_each_pid_task(pgrp, PIDTYPE_PGID, p);
  return ret; /* (sighing) "Often!" */
 }

@@ -234,23 +234,23 @@ int is_orphaned_pgrp(int pgrp)
  int retval;

  read_lock(&tasklist_lock);
- retval = will_become_orphaned_pgrp(pgrp, NULL);
+ retval = will_become_orphaned_pgrp(find_pid(pgrp), NULL);
  read_unlock(&tasklist_lock);

  return retval;
 }

-static int has_stopped_jobs(int pgrp)
+static int has_stopped_jobs(struct pid *pgrp)
 {
  int retval = 0;
  struct task_struct *p;
```

```
- do_each_task_pid(pgrp, PIDTYPE_PGID, p) {
+ do_each_pid_task(pgrp, PIDTYPE_PGID, p) {
   if (p->state != TASK_STOPPED)
    continue;
   retval = 1;
   break;
- } while_each_task_pid(pgrp, PIDTYPE_PGID, p);
+ } while_each_pid_task(pgrp, PIDTYPE_PGID, p);
  return retval;
 }

@@ -640,14 +640,14 @@ reparent_thread(struct task_struct *p, struct task_struct *father, int
traced)
   * than we are, and it was the only connection
   * outside, so the child pgrp is now orphaned.
   */
- if ((process_group(p) != process_group(father)) &&
-     (process_session(p) == process_session(father))) {
-  int pgrp = process_group(p);
+ if ((task_pgrp(p) != task_pgrp(father)) &&
+     (task_session(p) == task_session(father))) {
+  struct pid *pgrp = task_pgrp(p);

   if (will_become_orphaned_pgrp(pgrp, NULL) &&
       has_stopped_jobs(pgrp)) {
-   __kill_pg_info(SIGHUP, SEND_SIG_PRIV, pgrp);
-   __kill_pg_info(SIGCONT, SEND_SIG_PRIV, pgrp);
+   __kill_pgrp_info(SIGHUP, SEND_SIG_PRIV, pgrp);
+   __kill_pgrp_info(SIGCONT, SEND_SIG_PRIV, pgrp);
   }
  }
 }
@@ -727,6 +727,7 @@ static void exit_notify(struct task_struct *tsk)
  int state;
  struct task_struct *t;
  struct list_head ptrace_dead, *_p, *_n;
+ struct pid *pgrp;

  if (signal_pending(tsk) && !(tsk->signal->flags & SIGNAL_GROUP_EXIT)
      && !thread_group_empty(tsk)) {
@@ -779,12 +780,13 @@ static void exit_notify(struct task_struct *tsk)

  t = tsk->real_parent;

- if ((process_group(t) != process_group(tsk)) &&
-     (process_session(t) == process_session(tsk)) &&
-     will_become_orphaned_pgrp(process_group(tsk), tsk) &&
-     has_stopped_jobs(process_group(tsk))) {
```

```
-  __kill_pg_info(SIGHUP, SEND_SIG_PRIV, process_group(tsk));
-  __kill_pg_info(SIGCONT, SEND_SIG_PRIV, process_group(tsk));
+ pgrp = task_pgrp(tsk);
+ if ((task_pgrp(t) != pgrp) &&
+     (task_session(t) != task_session(tsk)) &&
+     will_become_orphaned_pgrp(pgrp, tsk) &&
+     has_stopped_jobs(pgrp)) {
+  __kill_pgrp_info(SIGHUP, SEND_SIG_PRIV, pgrp);
+  __kill_pgrp_info(SIGCONT, SEND_SIG_PRIV, pgrp);
  }

  /* Let father know we died
--
1.4.4.1.g278f
```


Subject: [PATCH 8/12] pid: Replace is_orphaned_pgrp with is_current_pgrp_orphaned

Posted by ebiederm on Wed, 13 Dec 2006 11:07:52 GMT

View Forum Message <> Reply to Message

Every call to is_orphaned_pgrp passed in process_group(current)
which is racy with respect to another thread changing our process
group. It didn't bite us because we were dealing with integers
and the worse we would get would be a stale answer.

In switching the checks to use struct pid to be a little more
efficient and prepare the way for pid namespaces this
race became apparent.

So I simplified the calls to the more specialized is_current_pgrp_orphaned
so I didn't have to worry about making logic changes to avoid the race.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
---
 drivers/char/n_tty.c  |    2 +-
 drivers/char/tty_io.c |    2 +-
 include/linux/tty.h   |    2 +-
 kernel/exit.c         |    4 ++--
 kernel/signal.c       |    2 +-
 5 files changed, 6 insertions(+), 6 deletions(-)

diff --git a/drivers/char/n_tty.c b/drivers/char/n_tty.c
```

```
index e96a00f..7f1ded8 100644
--- a/drivers/char/n_tty.c
+++ b/drivers/char/n_tty.c
@@ -1189,7 +1189,7 @@ static int job_control(struct tty_struct *tty, struct file *file)
    printk("read_chan: tty->pgrp <= 0!\n");
   else if (process_group(current) != tty->pgrp) {
    if (is_ignored(SIGTTIN) ||
-       is_orphaned_pgrp(process_group(current)))
+       is_current_pgrp_orphaned())
     return -EIO;
    kill_pg(process_group(current), SIGTTIN, 1);
    return -ERESTARTSYS;
diff --git a/drivers/char/tty_io.c b/drivers/char/tty_io.c
index 74f078a..d8fdf45 100644
--- a/drivers/char/tty_io.c
+++ b/drivers/char/tty_io.c
@@ -1118,7 +1118,7 @@ int tty_check_change(struct tty_struct * tty)
   return 0;
  if (is_ignored(SIGTTOU))
   return 0;
- if (is_orphaned_pgrp(process_group(current)))
+ if (is_current_pgrp_orphaned())
   return -EIO;
  (void) kill_pg(process_group(current), SIGTTOU, 1);
  return -ERESTARTSYS;
diff --git a/include/linux/tty.h b/include/linux/tty.h
index 1185bca..13a4918 100644
--- a/include/linux/tty.h
+++ b/include/linux/tty.h
@@ -283,7 +283,7 @@ extern int tty_read_raw_data(struct tty_struct *tty, unsigned char *bufp,
        int buflen);
 extern void tty_write_message(struct tty_struct *tty, char *msg);

-extern int is_orphaned_pgrp(int pgrp);
+extern int is_current_pgrp_orphaned(void);
 extern int is_ignored(int sig);
 extern int tty_signal(int sig, struct tty_struct *tty);
 extern void tty_hangup(struct tty_struct * tty);
diff --git a/kernel/exit.c b/kernel/exit.c
index 5f8455e..610917d 100644
--- a/kernel/exit.c
+++ b/kernel/exit.c
@@ -229,12 +229,12 @@ static int will_become_orphaned_pgrp(struct pid *pgrp, struct
task_struct *ignor
  return ret; /* (sighing) "Often!" */
 }

-int is_orphaned_pgrp(int pgrp)
```

```
+int is_current_pgrp_orphaned(void)
 {
  int retval;

  read_lock(&tasklist_lock);
- retval = will_become_orphaned_pgrp(find_pid(pgrp), NULL);
+ retval = will_become_orphaned_pgrp(task_pgrp(current), NULL);
  read_unlock(&tasklist_lock);

  return retval;
diff --git a/kernel/signal.c b/kernel/signal.c
index 1e34d32..91caafa 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -1908,7 +1908,7 @@ relock:

     /* signals can be posted during this window */

-    if (is_orphaned_pgrp(process_group(current)))
+    if (is_current_pgrp_orphaned())
      goto relock;

     spin_lock_irq(&current->sighand->siglock);
--
1.4.4.1.g278f
```

Subject: [PATCH 9/12] tty: Update the tty layer to work with struct pid.
Posted by ebiederm on Wed, 13 Dec 2006 11:07:53 GMT
View Forum Message <> Reply to Message

Of kernel subsystems that work with pids the tty layer
is probably the largest consumer. But it has the nice
virtue that the assiation with a session only lasts until
the session leader exits. Which means that no reference
counting is required. So using struct pid winds up
being a simple optimization to avoid hash table lookups.

In the long term the use of pid_nr also ensures that
when we have multiple pid spaces mixed everything will
work correctly.

Signed-off-by: Eric W. Biederman <eric@maxwell.lnxi.com>

```
---
 arch/um/drivers/line.c     |    2 +-
 drivers/char/ip2/ip2main.c |    4 +-
 drivers/char/n_tty.c       |   12 ++--
 drivers/char/tty_io.c      |  129 ++++++++++++++++++++++++++++++++-----------
 drivers/char/vt.c          |    4 +-
 fs/proc/array.c            |    2 +-
 include/linux/init_task.h  |    2 +-
 include/linux/sched.h      |    2 +-
 include/linux/tty.h        |    4 +-
 kernel/fork.c              |    2 +-
 kernel/sys.c               |    1 -
 11 files changed, 95 insertions(+), 69 deletions(-)

diff --git a/arch/um/drivers/line.c b/arch/um/drivers/line.c
index 83301e1..3c1a1d4 100644
--- a/arch/um/drivers/line.c
+++ b/arch/um/drivers/line.c
@@ -737,7 +737,7 @@ static irqreturn_t winch_interrupt(int irq, void *data)
   line = tty->driver_data;
   chan_window_size(&line->chan_list, &tty->winsize.ws_row,
      &tty->winsize.ws_col);
-  kill_pg(tty->pgrp, SIGWINCH, 1);
+  kill_pgrp(tty->pgrp, SIGWINCH, 1);
  }
  out:
  if(winch->fd != -1)
diff --git a/drivers/char/ip2/ip2main.c b/drivers/char/ip2/ip2main.c
index 7c70310..83c7258 100644
--- a/drivers/char/ip2/ip2main.c
+++ b/drivers/char/ip2/ip2main.c
@@ -1271,8 +1271,8 @@ static void do_input(struct work_struct *work)
 // code duplicated from n_tty (ldisc)
 static inline void  isig(int sig, struct tty_struct *tty, int flush)
 {
- if (tty->pgrp > 0)
-  kill_pg(tty->pgrp, sig, 1);
+ if (tty->pgrp)
+  kill_pgrp(tty->pgrp, sig, 1);
  if (flush || !L_NOFLSH(tty)) {
   if ( tty->ldisc.flush_buffer )
    tty->ldisc.flush_buffer(tty);
diff --git a/drivers/char/n_tty.c b/drivers/char/n_tty.c
index 7f1ded8..2b50a50 100644
--- a/drivers/char/n_tty.c
+++ b/drivers/char/n_tty.c
@@ -579,8 +579,8 @@ static void eraser(unsigned char c, struct tty_struct *tty)
```

```
 static inline void isig(int sig, struct tty_struct *tty, int flush)
 {
- if (tty->pgrp > 0)
-  kill_pg(tty->pgrp, sig, 1);
+ if (tty->pgrp)
+  kill_pgrp(tty->pgrp, sig, 1);
  if (flush || !L_NOFLSH(tty)) {
   n_tty_flush_buffer(tty);
   if (tty->driver->flush_buffer)
@@ -1185,13 +1185,13 @@ static int job_control(struct tty_struct *tty, struct file *file)
  /* don't stop on /dev/console */
  if (file->f_op->write != redirected_tty_write &&
      current->signal->tty == tty) {
-  if (tty->pgrp <= 0)
-   printk("read_chan: tty->pgrp <= 0!\n");
-  else if (process_group(current) != tty->pgrp) {
+  if (!tty->pgrp)
+   printk("read_chan: no tty->pgrp!\n");
+  else if (task_pgrp(current) != tty->pgrp) {
    if (is_ignored(SIGTTIN) ||
        is_current_pgrp_orphaned())
     return -EIO;
-   kill_pg(process_group(current), SIGTTIN, 1);
+   kill_pgrp(task_pgrp(current), SIGTTIN, 1);
    return -ERESTARTSYS;
   }
  }
diff --git a/drivers/char/tty_io.c b/drivers/char/tty_io.c
index d8fdf45..fe98f8c 100644
--- a/drivers/char/tty_io.c
+++ b/drivers/char/tty_io.c
@@ -155,7 +155,7 @@ int tty_ioctl(struct inode * inode, struct file * file,
       unsigned int cmd, unsigned long arg);
 static int tty_fasync(int fd, struct file * filp, int on);
 static void release_mem(struct tty_struct *tty, int idx);
-static void __proc_set_tty(struct task_struct *tsk, struct tty_struct *tty);
+static struct pid *__proc_set_tty(struct task_struct *tsk, struct tty_struct *tty);

 /**
  * alloc_tty_struct - allocate a tty object
@@ -1110,17 +1110,17 @@ int tty_check_change(struct tty_struct * tty)
 {
  if (current->signal->tty != tty)
   return 0;
- if (tty->pgrp <= 0) {
-  printk(KERN_WARNING "tty_check_change: tty->pgrp <= 0!\n");
+ if (!tty->pgrp) {
+  printk(KERN_WARNING "tty_check_change: tty->pgrp == NULL!\n");
```

```
   return 0;
  }
- if (process_group(current) == tty->pgrp)
+ if (task_pgrp(current) == tty->pgrp)
   return 0;
  if (is_ignored(SIGTTOU))
   return 0;
  if (is_current_pgrp_orphaned())
   return -EIO;
- (void) kill_pg(process_group(current), SIGTTOU, 1);
+ (void) kill_pgrp(task_pgrp(current), SIGTTOU, 1);
  return -ERESTARTSYS;
 }

@@ -1355,8 +1355,8 @@ static void do_tty_hangup(struct work_struct *work)
    tty_release is called */

  read_lock(&tasklist_lock);
- if (tty->session > 0) {
-  do_each_task_pid(tty->session, PIDTYPE_SID, p) {
+ if (tty->session) {
+  do_each_pid_task(tty->session, PIDTYPE_SID, p) {
    spin_lock_irq(&p->sighand->siglock);
    if (p->signal->tty == tty)
     p->signal->tty = NULL;
@@ -1366,16 +1366,17 @@ static void do_tty_hangup(struct work_struct *work)
    }
    __group_send_sig_info(SIGHUP, SEND_SIG_PRIV, p);
    __group_send_sig_info(SIGCONT, SEND_SIG_PRIV, p);
-   if (tty->pgrp > 0)
-    p->signal->tty_old_pgrp = tty->pgrp;
+   put_pid(p->signal->tty_old_pgrp);  /* A noop */
+   if (tty->pgrp)
+    p->signal->tty_old_pgrp = get_pid(tty->pgrp);
    spin_unlock_irq(&p->sighand->siglock);
-  } while_each_task_pid(tty->session, PIDTYPE_SID, p);
+  } while_each_pid_task(tty->session, PIDTYPE_SID, p);
  }
  read_unlock(&tasklist_lock);

  tty->flags = 0;
- tty->session = 0;
- tty->pgrp = -1;
+ tty->session = NULL;
+ tty->pgrp = NULL;
  tty->ctrl_status = 0;
  /*
   * If one of the devices matches a console pointer, we
```

```
@@ -1460,12 +1461,12 @@ int tty_hung_up_p(struct file * filp)

 EXPORT_SYMBOL(tty_hung_up_p);

-static void session_clear_tty(pid_t session)
+static void session_clear_tty(struct pid *session)
 {
  struct task_struct *p;
- do_each_task_pid(session, PIDTYPE_SID, p) {
+ do_each_pid_task(session, PIDTYPE_SID, p) {
   proc_clear_tty(p);
- } while_each_task_pid(session, PIDTYPE_SID, p);
+ } while_each_pid_task(session, PIDTYPE_SID, p);
 }

 /**
@@ -1495,48 +1496,54 @@ static void session_clear_tty(pid_t session)
 void disassociate_ctty(int on_exit)
 {
  struct tty_struct *tty;
- int tty_pgrp = -1;
+ struct pid *tty_pgrp = NULL;

  lock_kernel();

  mutex_lock(&tty_mutex);
  tty = get_current_tty();
  if (tty) {
-  tty_pgrp = tty->pgrp;
+  tty_pgrp = get_pid(tty->pgrp);
   mutex_unlock(&tty_mutex);
   /* XXX: here we race, there is nothing protecting tty */
   if (on_exit && tty->driver->type != TTY_DRIVER_TYPE_PTY)
    tty_vhangup(tty);
  } else if (on_exit) {
-  pid_t old_pgrp;
+  struct pid *old_pgrp;
   spin_lock_irq(&current->sighand->siglock);
   old_pgrp = current->signal->tty_old_pgrp;
-  current->signal->tty_old_pgrp = 0;
+  current->signal->tty_old_pgrp = NULL;
   spin_unlock_irq(&current->sighand->siglock);
   if (old_pgrp) {
-   kill_pg(old_pgrp, SIGHUP, on_exit);
-   kill_pg(old_pgrp, SIGCONT, on_exit);
+   kill_pgrp(old_pgrp, SIGHUP, on_exit);
+   kill_pgrp(old_pgrp, SIGCONT, on_exit);
+   put_pid(old_pgrp);
```

```
   }
   mutex_unlock(&tty_mutex);
   unlock_kernel();
   return;
  }
- if (tty_pgrp > 0) {
-  kill_pg(tty_pgrp, SIGHUP, on_exit);
+ if (tty_pgrp) {
+  kill_pgrp(tty_pgrp, SIGHUP, on_exit);
   if (!on_exit)
-   kill_pg(tty_pgrp, SIGCONT, on_exit);
+   kill_pgrp(tty_pgrp, SIGCONT, on_exit);
+  put_pid(tty_pgrp);
  }

  spin_lock_irq(&current->sighand->siglock);
+ tty_pgrp = current->signal->tty_old_pgrp;
  current->signal->tty_old_pgrp = 0;
  spin_unlock_irq(&current->sighand->siglock);
+ put_pid(tty_pgrp);

  mutex_lock(&tty_mutex);
  /* It is possible that do_tty_hangup has free'd this tty */
  tty = get_current_tty();
  if (tty) {
-  tty->session = 0;
-  tty->pgrp = 0;
+  put_pid(tty->session);
+  put_pid(tty->pgrp);
+  tty->session = NULL;
+  tty->pgrp = NULL;
  } else {
 #ifdef TTY_DEBUG_HANGUP
   printk(KERN_DEBUG "error attempted to write to tty [0x%p]"
@@ -1547,7 +1554,7 @@ void disassociate_ctty(int on_exit)

  /* Now clear signal->tty under the lock */
  read_lock(&tasklist_lock);
- session_clear_tty(process_session(current));
+ session_clear_tty(task_session(current));
  read_unlock(&tasklist_lock);
  unlock_kernel();
 }
@@ -2484,6 +2491,7 @@ static int tty_open(struct inode * inode, struct file * filp)
  int index;
  dev_t device = inode->i_rdev;
  unsigned short saved_flags = filp->f_flags;
+ struct pid *old_pgrp;
```

```
  nonseekable_open(inode, filp);

@@ -2577,15 +2585,17 @@ got_driver:
    goto retry_open;
   }

+ old_pgrp = NULL;
   mutex_lock(&tty_mutex);
   spin_lock_irq(&current->sighand->siglock);
   if (!noctty &&
       current->signal->leader &&
       !current->signal->tty &&
-      tty->session == 0)
-   __proc_set_tty(current, tty);
+      tty->session == NULL)
+   old_pgrp = __proc_set_tty(current, tty);
   spin_unlock_irq(&current->sighand->siglock);
   mutex_unlock(&tty_mutex);
+ put_pid(old_pgrp);
   return 0;
  }

@@ -2724,9 +2734,18 @@ static int tty_fasync(int fd, struct file * filp, int on)
    return retval;

   if (on) {
+   enum pid_type type;
+   struct pid *pid;
    if (!waitqueue_active(&tty->read_wait))
     tty->minimum_to_wake = 1;
-   retval = f_setown(filp, (-tty->pgrp) ? : current->pid, 0);
+   if (tty->pgrp) {
+    pid = tty->pgrp;
+    type = PIDTYPE_PGID;
+   } else {
+    pid = task_pid(current);
+    type = PIDTYPE_PID;
+   }
+   retval = __f_setown(filp, pid, type, 0);
    if (retval)
     return retval;
   } else {
@@ -2828,10 +2847,10 @@ static int tiocswinsz(struct tty_struct *tty, struct tty_struct *real_tty,
    }
   }
  #endif
- if (tty->pgrp > 0)
```

```
-  kill_pg(tty->pgrp, SIGWINCH, 1);
- if ((real_tty->pgrp != tty->pgrp) && (real_tty->pgrp > 0))
-  kill_pg(real_tty->pgrp, SIGWINCH, 1);
+ if (tty->pgrp)
+  kill_pgrp(tty->pgrp, SIGWINCH, 1);
+ if ((real_tty->pgrp != tty->pgrp) && real_tty->pgrp)
+  kill_pgrp(real_tty->pgrp, SIGWINCH, 1);
  tty->winsize = tmp_ws;
  real_tty->winsize = tmp_ws;
 done:
@@ -2916,8 +2935,7 @@ static int fionbio(struct file *file, int __user *p)
 static int tiocsctty(struct tty_struct *tty, int arg)
 {
  int ret = 0;
- if (current->signal->leader &&
-   (process_session(current) == tty->session))
+ if (current->signal->leader && (task_session(current) == tty->session))
   return ret;

  mutex_lock(&tty_mutex);
@@ -2930,7 +2948,7 @@ static int tiocsctty(struct tty_struct *tty, int arg)
   goto unlock;
  }

- if (tty->session > 0) {
+ if (tty->session) {
   /*
    * This tty is already the controlling
    * tty for another session group!
@@ -2973,7 +2991,7 @@ static int tiocgpgrp(struct tty_struct *tty, struct tty_struct *real_tty, pid_t
   */
  if (tty == real_tty && current->signal->tty != real_tty)
   return -ENOTTY;
- return put_user(real_tty->pgrp, p);
+ return put_user(pid_nr(real_tty->pgrp), p);
 }

 /**
@@ -3000,7 +3018,7 @@ static int tiocspgrp(struct tty_struct *tty, struct tty_struct *real_tty, pid_t
   return retval;
  if (!current->signal->tty ||
      (current->signal->tty != real_tty) ||
-     (real_tty->session != process_session(current)))
+     (real_tty->session != task_session(current)))
   return -ENOTTY;
  if (get_user(pgrp_nr, p))
   return -EFAULT;
@@ -3015,7 +3033,8 @@ static int tiocspgrp(struct tty_struct *tty, struct tty_struct *real_tty, pid_t
```

```
  if (session_of_pgrp(pgrp) != task_session(current))
   goto out_unlock;
  retval = 0;
- real_tty->pgrp = pgrp_nr;
+ put_pid(real_tty->pgrp);
+ real_tty->pgrp = get_pid(pgrp);
 out_unlock:
  rcu_read_unlock();
  return retval;
@@ -3041,9 +3060,9 @@ static int tiocgsid(struct tty_struct *tty, struct tty_struct *real_tty, pid_t _
  */
  if (tty == real_tty && current->signal->tty != real_tty)
   return -ENOTTY;
- if (real_tty->session <= 0)
+ if (!real_tty->session)
   return -ENOTTY;
- return put_user(real_tty->session, p);
+ return put_user(pid_nr(real_tty->session), p);
 }

 /**
@@ -3343,7 +3362,7 @@ void __do_SAK(struct tty_struct *tty)
  tty_hangup(tty);
 #else
  struct task_struct *g, *p;
- int session;
+ struct pid *session;
  int  i;
  struct file *filp;
  struct tty_ldisc *disc;
@@ -3364,12 +3383,12 @@ void __do_SAK(struct tty_struct *tty)

  read_lock(&tasklist_lock);
  /* Kill the entire session */
- do_each_task_pid(session, PIDTYPE_SID, p) {
+ do_each_pid_task(session, PIDTYPE_SID, p) {
   printk(KERN_NOTICE "SAK: killed process %d"
    " (%s): process_session(p)==tty->session\n",
    p->pid, p->comm);
   send_sig(SIGKILL, p, 1);
- } while_each_task_pid(session, PIDTYPE_SID, p);
+ } while_each_pid_task(session, PIDTYPE_SID, p);
  /* Now kill any processes that happen to have the
   * tty open.
   */
@@ -3538,7 +3557,8 @@ static void initialize_tty_struct(struct tty_struct *tty)
  memset(tty, 0, sizeof(struct tty_struct));
  tty->magic = TTY_MAGIC;
```

```
  tty_ldisc_assign(tty, tty_ldisc_get(N_TTY));
- tty->pgrp = -1;
+ tty->session = NULL;
+ tty->pgrp = NULL;
  tty->overrun_time = jiffies;
  tty->buf.head = tty->buf.tail = NULL;
  tty_buffer_init(tty);
@@ -3809,21 +3829,28 @@ void proc_clear_tty(struct task_struct *p)
 }
 EXPORT_SYMBOL(proc_clear_tty);

-static void __proc_set_tty(struct task_struct *tsk, struct tty_struct *tty)
+static struct pid *__proc_set_tty(struct task_struct *tsk, struct tty_struct *tty)
 {
+ struct pid *old_pgrp;
  if (tty) {
-  tty->session = process_session(tsk);
-  tty->pgrp = process_group(tsk);
+  tty->session = get_pid(task_session(tsk));
+  tty->pgrp = get_pid(task_pgrp(tsk));
  }
+ old_pgrp = tsk->signal->tty_old_pgrp;
  tsk->signal->tty = tty;
- tsk->signal->tty_old_pgrp = 0;
+ tsk->signal->tty_old_pgrp = NULL;
+ return old_pgrp;
 }

 void proc_set_tty(struct task_struct *tsk, struct tty_struct *tty)
 {
+ struct pid *old_pgrp;
+
  spin_lock_irq(&tsk->sighand->siglock);
- __proc_set_tty(tsk, tty);
+ old_pgrp = __proc_set_tty(tsk, tty);
  spin_unlock_irq(&tsk->sighand->siglock);
+
+ put_pid(old_pgrp);
 }

 struct tty_struct *get_current_tty(void)
diff --git a/drivers/char/vt.c b/drivers/char/vt.c
index a8239da..4d52cab 100644
--- a/drivers/char/vt.c
+++ b/drivers/char/vt.c
@@ -869,8 +869,8 @@ int vc_resize(struct vc_data *vc, unsigned int cols, unsigned int lines)
   ws.ws_col = vc->vc_cols;
   ws.ws_ypixel = vc->vc_scan_lines;
```

```
   if ((ws.ws_row != cws->ws_row || ws.ws_col != cws->ws_col) &&
-      vc->vc_tty->pgrp > 0)
-   kill_pg(vc->vc_tty->pgrp, SIGWINCH, 1);
+      vc->vc_tty->pgrp)
+   kill_pgrp(vc->vc_tty->pgrp, SIGWINCH, 1);
   *cws = ws;
  }

diff --git a/fs/proc/array.c b/fs/proc/array.c
index 70e4fab..07c9cdb 100644
--- a/fs/proc/array.c
+++ b/fs/proc/array.c
@@ -351,7 +351,7 @@ static int do_task_stat(struct task_struct *task, char * buffer, int whole)
   struct signal_struct *sig = task->signal;

   if (sig->tty) {
-   tty_pgrp = sig->tty->pgrp;
+   tty_pgrp = pid_nr(sig->tty->pgrp);
    tty_nr = new_encode_dev(tty_devnum(sig->tty));
   }

diff --git a/include/linux/init_task.h b/include/linux/init_task.h
index b531515..9991297 100644
--- a/include/linux/init_task.h
+++ b/include/linux/init_task.h
@@ -66,7 +66,7 @@
  .cpu_timers = INIT_CPU_TIMERS(sig.cpu_timers),  \
  .rlim  = INIT_RLIMITS,     \
  .pgrp  = 1,      \
- .tty_old_pgrp   = 0,      \
+ .tty_old_pgrp   = NULL,      \
  { .__session      = 1},      \
 }

diff --git a/include/linux/sched.h b/include/linux/sched.h
index ea92e5c..ef0d6fe 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -436,7 +436,7 @@ struct signal_struct {

  /* job control IDs */
  pid_t pgrp;
- pid_t tty_old_pgrp;
+ struct pid *tty_old_pgrp;

  union {
   pid_t session __deprecated;
diff --git a/include/linux/tty.h b/include/linux/tty.h
```

```
index 13a4918..f07e390 100644
--- a/include/linux/tty.h
+++ b/include/linux/tty.h
@@ -177,8 +177,8 @@ struct tty_struct {
  struct mutex termios_mutex;
  struct ktermios *termios, *termios_locked;
  char name[64];
- int pgrp;
- int session;
+ struct pid *pgrp;
+ struct pid *session;
  unsigned long flags;
  int count;
  struct winsize winsize;
diff --git a/kernel/fork.c b/kernel/fork.c
index d16c566..24b4af0 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -869,7 +869,7 @@ static inline int copy_signal(unsigned long clone_flags, struct task_struct
* ts
  sig->it_prof_incr = cputime_zero;

  sig->leader = 0; /* session leadership doesn't inherit */
- sig->tty_old_pgrp = 0;
+ sig->tty_old_pgrp = NULL;

  sig->utime = sig->stime = sig->cutime = sig->cstime = cputime_zero;
  sig->nvcsw = sig->nivcsw = sig->cnvcsw = sig->cnivcsw = 0;
diff --git a/kernel/sys.c b/kernel/sys.c
index c7675c1..a8f2ebe 100644
--- a/kernel/sys.c
+++ b/kernel/sys.c
@@ -1503,7 +1503,6 @@ asmlinkage long sys_setsid(void)

  spin_lock(&group_leader->sighand->siglock);
  group_leader->signal->tty = NULL;
- group_leader->signal->tty_old_pgrp = 0;
  spin_unlock(&group_leader->sighand->siglock);

  err = process_group(group_leader);
--
1.4.4.1.g278f
```

Subject: [PATCH 10/12] pid: Replace do/while_each_task_pid with do/while_each_pid_task
Posted by ebiederm on Wed, 13 Dec 2006 11:07:54 GMT
View Forum Message <> Reply to Message

There isn't any real advantage to this change except that
it allows the old functions to be removed. Which is easier
on maintenance and puts the code in a more uniform style.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/ioprio.c | 18 ++++++++++------
 kernel/capability.c | 8 +++++---
 kernel/sys.c | 40 ++++++++++++++++++++++++++++++++++++----------
 3 files changed, 41 insertions(+), 25 deletions(-)

```
diff --git a/fs/ioprio.c b/fs/ioprio.c
index 89e8da1..10d2c21 100644
--- a/fs/ioprio.c
+++ b/fs/ioprio.c
@@ -60,6 +60,7 @@ asmlinkage long sys_ioprio_set(int which, int who, int ioprio)
  int data = IOPRIO_PRIO_DATA(ioprio);
  struct task_struct *p, *g;
  struct user_struct *user;
+ struct pid *pgrp;
  int ret;

  switch (class) {
@@ -98,12 +99,14 @@ asmlinkage long sys_ioprio_set(int which, int who, int ioprio)
    break;
   case IOPRIO_WHO_PGRP:
    if (!who)
-    who = process_group(current);
-   do_each_task_pid(who, PIDTYPE_PGID, p) {
+    pgrp = task_pgrp(current);
+   else
+    pgrp = find_pid(who);
+   do_each_pid_task(pgrp, PIDTYPE_PGID, p) {
     ret = set_task_ioprio(p, ioprio);
     if (ret)
      break;
-   } while_each_task_pid(who, PIDTYPE_PGID, p);
+   } while_each_pid_task(pgrp, PIDTYPE_PGID, p);
    break;
   case IOPRIO_WHO_USER:
    if (!who)
@@ -167,6 +170,7 @@ asmlinkage long sys_ioprio_get(int which, int who)
 {
  struct task_struct *g, *p;
```

```
  struct user_struct *user;
+ struct pid *pgrp;
  int ret = -ESRCH;
  int tmpio;

@@ -182,8 +186,10 @@ asmlinkage long sys_ioprio_get(int which, int who)
     break;
    case IOPRIO_WHO_PGRP:
     if (!who)
-     who = process_group(current);
-    do_each_task_pid(who, PIDTYPE_PGID, p) {
+     pgrp = task_pgrp(current);
+    else
+     pgrp = find_pid(who);
+    do_each_pid_task(pgrp, PIDTYPE_PGID, p) {
      tmpio = get_task_ioprio(p);
      if (tmpio < 0)
       continue;
@@ -191,7 +197,7 @@ asmlinkage long sys_ioprio_get(int which, int who)
       ret = tmpio;
      else
       ret = ioprio_best(ret, tmpio);
-    } while_each_task_pid(who, PIDTYPE_PGID, p);
+    } while_each_pid_task(pgrp, PIDTYPE_PGID, p);
     break;
    case IOPRIO_WHO_USER:
     if (!who)
diff --git a/kernel/capability.c b/kernel/capability.c
index edb845a..c8d3c77 100644
--- a/kernel/capability.c
+++ b/kernel/capability.c
@@ -92,15 +92,17 @@ out:
  * cap_set_pg - set capabilities for all processes in a given process
  * group.  We call this holding task_capability_lock and tasklist_lock.
  */
-static inline int cap_set_pg(int pgrp, kernel_cap_t *effective,
+static inline int cap_set_pg(int pgrp_nr, kernel_cap_t *effective,
          kernel_cap_t *inheritable,
          kernel_cap_t *permitted)
 {
  struct task_struct *g, *target;
  int ret = -EPERM;
  int found = 0;
+ struct pid *pgrp;

- do_each_task_pid(pgrp, PIDTYPE_PGID, g) {
+ pgrp = find_pid(pgrp_nr);
+ do_each_pid_task(pgrp, PIDTYPE_PGID, g) {
```

```
   target = g;
   while_each_thread(g, target) {
    if (!security_capset_check(target, effective,
@@ -113,7 +115,7 @@ static inline int cap_set_pg(int pgrp, kernel_cap_t *effective,
    }
    found = 1;
   }
- } while_each_task_pid(pgrp, PIDTYPE_PGID, g);
+ } while_each_pid_task(pgrp, PIDTYPE_PGID, g);

  if (!found)
       ret = 0;
diff --git a/kernel/sys.c b/kernel/sys.c
index a8f2ebe..c452ba9 100644
--- a/kernel/sys.c
+++ b/kernel/sys.c
@@ -589,6 +589,7 @@ asmlinkage long sys_setpriority(int which, int who, int niceval)
  struct task_struct *g, *p;
  struct user_struct *user;
  int error = -EINVAL;
+ struct pid *pgrp;

  if (which > 2 || which < 0)
   goto out;
@@ -603,18 +604,21 @@ asmlinkage long sys_setpriority(int which, int who, int niceval)
  read_lock(&tasklist_lock);
  switch (which) {
   case PRIO_PROCESS:
-   if (!who)
-    who = current->pid;
-   p = find_task_by_pid(who);
+   if (who)
+    p = find_task_by_pid(who);
+   else
+    p = current;
    if (p)
     error = set_one_prio(p, niceval, error);
    break;
   case PRIO_PGRP:
-   if (!who)
-    who = process_group(current);
-   do_each_task_pid(who, PIDTYPE_PGID, p) {
+   if (who)
+    pgrp = find_pid(who);
+   else
+    pgrp = task_pgrp(current);
+   do_each_pid_task(pgrp, PIDTYPE_PGID, p) {
     error = set_one_prio(p, niceval, error);
```

```
-   } while_each_task_pid(who, PIDTYPE_PGID, p);
+   } while_each_pid_task(pgrp, PIDTYPE_PGID, p);
     break;
    case PRIO_USER:
     user = current->user;
@@ -649,6 +653,7 @@ asmlinkage long sys_getpriority(int which, int who)
  struct task_struct *g, *p;
  struct user_struct *user;
  long niceval, retval = -ESRCH;
+ struct pid *pgrp;

  if (which > 2 || which < 0)
   return -EINVAL;
@@ -656,9 +661,10 @@ asmlinkage long sys_getpriority(int which, int who)
  read_lock(&tasklist_lock);
  switch (which) {
   case PRIO_PROCESS:
-   if (!who)
-    who = current->pid;
-   p = find_task_by_pid(who);
+   if (who)
+    p = find_task_by_pid(who);
+   else
+    p = current;
    if (p) {
     niceval = 20 - task_nice(p);
     if (niceval > retval)
@@ -666,13 +672,15 @@ asmlinkage long sys_getpriority(int which, int who)
    }
    break;
   case PRIO_PGRP:
-   if (!who)
-    who = process_group(current);
-   do_each_task_pid(who, PIDTYPE_PGID, p) {
+   if (who)
+    pgrp = find_pid(who);
+   else
+    pgrp = task_pgrp(current);
+   do_each_pid_task(pgrp, PIDTYPE_PGID, p) {
     niceval = 20 - task_nice(p);
     if (niceval > retval)
      retval = niceval;
-   } while_each_task_pid(who, PIDTYPE_PGID, p);
+   } while_each_pid_task(pgrp, PIDTYPE_PGID, p);
    break;
   case PRIO_USER:
    user = current->user;
@@ -1381,7 +1389,7 @@ asmlinkage long sys_setpgid(pid_t pid, pid_t pgid)
```

```
  if (p->real_parent == group_leader) {
   err = -EPERM;
-  if (process_session(p) != process_session(group_leader))
+  if (task_session(p) != task_session(group_leader))
    goto out;
   err = -EACCES;
   if (p->did_exec)
@@ -1400,7 +1408,7 @@ asmlinkage long sys_setpgid(pid_t pid, pid_t pgid)
   struct task_struct *g =
    find_task_by_pid_type(PIDTYPE_PGID, pgid);

-  if (!g || process_session(g) != process_session(group_leader))
+  if (!g || task_session(g) != task_session(group_leader))
    goto out;
  }

--
1.4.4.1.g278f
```

Subject: [PATCH 11/12] pid: Remove now unused do_each_task_pid and while_each_task_pid

Posted by ebiederm on Wed, 13 Dec 2006 11:07:55 GMT

View Forum Message <> Reply to Message

Now that I have changed all of the users remove the old version of these functions. This should be a clear hint to any out of tree users that they should use do_each_pid_task and while_each_pid_task for new code.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
---
 include/linux/pid.h |   14 --------------
 1 files changed, 0 insertions(+), 14 deletions(-)

diff --git a/include/linux/pid.h b/include/linux/pid.h
index 4dec047..2ac27f9 100644
--- a/include/linux/pid.h
+++ b/include/linux/pid.h
@@ -105,20 +105,6 @@ static inline pid_t pid_nr(struct pid *pid)
  return nr;
 }
```

```
-
-#define do_each_task_pid(who, type, task)    \
- do {        \
-  struct hlist_node *pos___;    \
-  struct pid *pid___ = find_pid(who);   \
-  if (pid___ != NULL)     \
-   hlist_for_each_entry_rcu((task), pos___, \
-    &pid___->tasks[type], pids[type].node) {
-
-#define while_each_task_pid(who, type, task)    \
-   }      \
- } while (0)
-
-
 #define do_each_pid_task(pid, type, task)    \
  do {        \
   struct hlist_node *pos___;    \
--
1.4.4.1.g278f
```

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

Subject: [PATCH 12/12] pid: Remove the now unused kill_pg kill_pg_info and __kill_pg_info

Posted by ebiederm on Wed, 13 Dec 2006 11:07:56 GMT
View Forum Message <> Reply to Message

Now that I have changed all of the in-tree users remove the old
version of these functions. This should make it clear to any
out of tree users that they should be using kill_pgrp kill_pgrp_info
or __kill_pgrp_info instead.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 include/linux/sched.h | 3 ---
 kernel/signal.c | 27 ---------------------------
 2 files changed, 0 insertions(+), 30 deletions(-)

diff --git a/include/linux/sched.h b/include/linux/sched.h
index ef0d6fe..f455f02 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1328,14 +1328,11 @@ extern int kill_pid_info(int sig, struct siginfo *info, struct pid *pid);

```
 extern int kill_pid_info_as_uid(int, struct siginfo *, struct pid *, uid_t, uid_t, u32);
 extern int kill_pgrp(struct pid *pid, int sig, int priv);
 extern int kill_pid(struct pid *pid, int sig, int priv);
-extern int __kill_pg_info(int sig, struct siginfo *info, pid_t pgrp);
-extern int kill_pg_info(int, struct siginfo *, pid_t);
 extern void do_notify_parent(struct task_struct *, int);
 extern void force_sig(int, struct task_struct *);
 extern void force_sig_specific(int, struct task_struct *);
 extern int send_sig(int, struct task_struct *, int);
 extern void zap_other_threads(struct task_struct *p);
-extern int kill_pg(pid_t, int, int);
 extern int kill_proc(pid_t, int, int);
 extern struct sigqueue *sigqueue_alloc(void);
 extern void sigqueue_free(struct sigqueue *);
diff --git a/kernel/signal.c b/kernel/signal.c
index 91caafa..2b29e95 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -1096,26 +1096,6 @@ int kill_pgrp_info(int sig, struct siginfo *info, struct pid *pgrp)
  return retval;
 }

-int __kill_pg_info(int sig, struct siginfo *info, pid_t pgrp)
-{
- if (pgrp <= 0)
-  return -EINVAL;
-
- return __kill_pgrp_info(sig, info, find_pid(pgrp));
-}
-
-int
-kill_pg_info(int sig, struct siginfo *info, pid_t pgrp)
-{
- int retval;
-
- read_lock(&tasklist_lock);
- retval = __kill_pg_info(sig, info, pgrp);
- read_unlock(&tasklist_lock);
-
- return retval;
-}
-
 int kill_pid_info(int sig, struct siginfo *info, struct pid *pid)
 {
  int error;
@@ -1316,12 +1296,6 @@ int kill_pid(struct pid *pid, int sig, int priv)
 EXPORT_SYMBOL(kill_pid);
```

```
 int
-kill_pg(pid_t pgrp, int sig, int priv)
-{
- return kill_pg_info(sig, __si_special(priv), pgrp);
-}
-
-int
 kill_proc(pid_t pid, int sig, int priv)
 {
  return kill_proc_info(sig, __si_special(priv), pid);
@@ -1958,7 +1932,6 @@ EXPORT_SYMBOL(recalc_sigpending);
 EXPORT_SYMBOL_GPL(dequeue_signal);
 EXPORT_SYMBOL(flush_signals);
 EXPORT_SYMBOL(force_sig);
-EXPORT_SYMBOL(kill_pg);
 EXPORT_SYMBOL(kill_proc);
 EXPORT_SYMBOL(ptrace_notify);
 EXPORT_SYMBOL(send_sig);
--
1.4.4.1.g278f
```

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

Subject: Re: [PATCH 0/12] tty layer and misc struct pid conversions
Posted by Andrew Morton on Wed, 13 Dec 2006 22:10:36 GMT
View Forum Message <> Reply to Message

On Wed, 13 Dec 2006 04:03:39 -0700
ebiederm@xmission.com (Eric W. Biederman) wrote:

> The aim of this patch set is to start wrapping up the struct pid
> conversions.

hm, it touches a lot of tricky code which few people are familar
with. Worried.

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

Subject: Re: [PATCH 0/12] tty layer and misc struct pid conversions
Posted by ebiederm on Thu, 14 Dec 2006 00:54:54 GMT

Andrew Morton <akpm@osdl.org> writes:

> On Wed, 13 Dec 2006 04:03:39 -0700
> ebiederm@xmission.com (Eric W. Biederman) wrote:
>
>> The aim of this patch set is to start wrapping up the struct pid
>> conversions.
>
> hm, it touches a lot of tricky code which few people are familar
> with. Worried.

Reasonable concern. The good thing is that the only big change was that
struct pid is reference counted while old style pids are not. Which
means that most of the pieces are simple substitutions. Although I admit
checking that the reference is correct especially in the tty layer
is tricky.

Eric

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