

---

Subject: [PATCH 07/17] Pid-NS(V3) Move alloc\_pid call to copy\_process  
Posted by [Sukadev Bhattiprolu](#) on Sat, 16 Jun 2007 23:01:05 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Subject: [PATCH 07/17] Pid-NS(V3) Move alloc\_pid call to copy\_process

From: Sukadev Bhattiprolu <[sukadev@us.ibm.com](mailto:sukadev@us.ibm.com)>

Move alloc\_pid() into copy\_process(). This will keep all pid and pid namespace code together and simplify error handling when we support multiple pid namespaces.

Changelog:

- [Eric Biederman] Move the check of copy\_process\_type to alloc\_pid()/free\_pid() and to avoid clutter in copy\_process().

Signed-off-by: Sukadev Bhattiprolu <[sukadev@us.ibm.com](mailto:sukadev@us.ibm.com)>

---

```
include/linux/pid.h |  7 ++++++-
kernel/fork.c      | 21 ++++++-----+
kernel/pid.c       | 10 ++++++++-+
3 files changed, 28 insertions(+), 10 deletions(-)
```

Index: lx26-22-rc4-mm2/include/linux/pid.h

```
=====
--- lx26-22-rc4-mm2.orig/include/linux/pid.h 2007-06-14 11:33:16.000000000 -0700
+++ lx26-22-rc4-mm2/include/linux/pid.h 2007-06-14 11:47:13.000000000 -0700
@@ -3,6 +3,11 @@
```

```
#include <linux/rcupdate.h>

+enum copy_process_type {
+ COPY_NON_IDLE_PROCESS,
+ COPY_IDLE_PROCESS,
+};
+
enum pid_type
{
 PIDTYPE_PID,
@@ -95,7 +100,7 @@ extern struct pid *FASTCALL(find_pid(int
 extern struct pid *find_get_pid(int nr);
 extern struct pid *find_ge_pid(int nr);

-extern struct pid *alloc_pid(void);
+extern struct pid *alloc_pid(enum copy_process_type);
 extern void FASTCALL(free_pid(struct pid *pid));

 static inline pid_t pid_to_nr(struct pid *pid)
```

Index: lx26-22-rc4-mm2/kernel/fork.c

```
=====
--- lx26-22-rc4-mm2.orig/kernel/fork.c 2007-06-14 11:33:16.000000000 -0700
+++ lx26-22-rc4-mm2/kernel/fork.c 2007-06-14 11:47:13.000000000 -0700
@@ -961,11 +961,12 @@ static struct task_struct *copy_process(
        unsigned long stack_size,
        int __user *parent_tidptr,
        int __user *child_tidptr,
-       struct pid *pid)
+       enum copy_process_type copy_src)
{
    int retval;
    struct task_struct *p = NULL;
    int container_callbacks_done = 0;
+   struct pid *pid;

    if ((clone_flags & (CLONE_NEWNS|CLONE_FS)) == (CLONE_NEWNS|CLONE_FS))
        return ERR_PTR(-EINVAL);
@@ -1026,6 +1027,10 @@ static struct task_struct *copy_process(
    if (p->binfo && !try_module_get(p->binfo->module))
        goto bad_fork_cleanup_put_domain;

+   pid = alloc_pid(copy_src);
+   if (!pid)
+       goto bad_fork_put_binfo_module;
+
    p->did_exec = 0;
    delayacct_tsk_init(p); /* Must remain after dup_task_struct() */
    copy_flags(clone_flags, p);
@@ -1313,6 +1318,8 @@ bad_fork_cleanup_container:
#endif
    container_exit(p, container_callbacks_done);
    delayacct_tsk_free(p);
+   free_pid(pid);
+bad_fork_put_binfo_module:
    if (p->binfo)
        module_put(p->binfo->module);
    bad_fork_cleanup_put_domain;
@@ -1339,7 +1346,7 @@ struct task_struct * __cpuinit fork_idle
    struct pt_regs regs;

    task = copy_process(CLONE_VM, 0, idle_regs(&regs), 0, NULL, NULL,
-   &init_struct_pid);
+   COPY_IDLE_PROCESS);
    if (!IS_ERR(task))
        init_idle(task, cpu);

@@ -1377,19 +1384,16 @@ long do_fork(unsigned long clone_flags,
```

```

{
    struct task_struct *p;
    int trace = 0;
- struct pid *pid = alloc_pid();
    long nr;

- if (!pid)
- return -EAGAIN;
- nr = pid->nr;
if (unlikely(current->ptrace)) {
    trace = fork_traceflag (clone_flags);
    if (trace)
        clone_flags |= CLONE_PTRACE;
}

- p = copy_process(clone_flags, stack_start, regs, stack_size, parent_tidptr, child_tidptr, pid);
+ p = copy_process(clone_flags, stack_start, regs, stack_size,
+ parent_tidptr, child_tidptr, COPY_NON_IDLE_PROCESS);
/*
 * Do this prior waking up the new thread - the thread pointer
 * might get invalid after that point, if the thread exits quickly.
@@ -1397,6 +1401,8 @@ long do_fork(unsigned long clone_flags,
if (!IS_ERR(p)) {
    struct completion vfork;

+ nr = pid_to_nr(task_pid(p));
+
    if (clone_flags & CLONE_VFORK) {
        p->vfork_done = &vfork;
        init_completion(&vfork);
@@ -1430,7 +1436,6 @@ long do_fork(unsigned long clone_flags,
    }
}
} else {
- free_pid(pid);
    nr = PTR_ERR(p);
}
return nr;
Index: lx26-22-rc4-mm2/kernel/pid.c
=====
--- lx26-22-rc4-mm2.orig/kernel/pid.c 2007-06-14 11:46:25.000000000 -0700
+++ lx26-22-rc4-mm2/kernel/pid.c 2007-06-14 11:47:13.000000000 -0700
@@ -200,6 +200,10 @@ fastcall void free_pid(struct pid *pid)
/* We can be called with write_lock_irq(&tasklist_lock) held */
    unsigned long flags;

+ /* check this here to keep copy_process() cleaner */
+ if (unlikely(pid == &init_struct_pid))

```

```

+ return;
+
 spin_lock_irqsave(&pidmap_lock, flags);
 hlist_del_rcu(&pid->pid_chain);
 spin_unlock_irqrestore(&pidmap_lock, flags);
@@ -208,12 +212,16 @@ fastcall void free_pid(struct pid *pid)
 call_rcu(&pid->rcu, delayed_put_pid);
}

-struct pid *alloc_pid(void)
+struct pid *alloc_pid(enum copy_process_type copy_src)
{
 struct pid *pid;
 enum pid_type type;
 int nr = -1;

+ /* check this here to keep copy_process() cleaner */
+ if (unlikely(copy_src == COPY_IDLE_PROCESS))
+ return &init_struct_pid;
+
 pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
 if (!pid)
 goto out;

--
```

---

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

---