Subject: [PATCH 8/20] Make alloc_pid(), free_pid() and put_pid() work with struct upid
Posted by Pavel Emelianov on Tue, 07 Aug 2007 09:29:50 GMT
View Forum Message <> Reply to Message

Each struct upid element of struct pid has to be initialized properly,
i.e. its nr mst be allocated from appropriate pidmap and ns set to
appropriate namespace.

When allocating a new pid, we need to know the namespace this pid will
live in, so the additional argument is added to alloc_pid().

On the other hand, the rest of the kernel still uses the pid->nr and
pid->pid_chain fields, so these ones are still initialized, but this
will be removed soon.

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

```
 include/linux/pid.h |    2 +-
 kernel/fork.c       |    2 +-
 kernel/pid.c        |   49 ++++++++++++++++++++++++++++++++++---------------
 3 files changed, 36 insertions(+), 17 deletions(-)

--- ./include/linux/pid.h.ve8 2007-08-06 15:39:05.000000000 +0400
+++ ./include/linux/pid.h 2007-08-06 15:39:05.000000000 +0400
@@ -111,7 +111,7 @@ 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(struct pid_namespace *ns);
 extern void FASTCALL(free_pid(struct pid *pid));

 static inline pid_t pid_nr(struct pid *pid)
--- ./kernel/fork.c.ve8 2007-08-06 15:39:02.000000000 +0400
+++ ./kernel/fork.c 2007-08-06 15:39:05.000000000 +0400
@@ -1029,7 +1029,7 @@ static struct task_struct *copy_process(
   goto bad_fork_cleanup_put_domain;

  if (pid != &init_struct_pid) {
-  pid = alloc_pid();
+  pid = alloc_pid(task_active_pid_ns(p));
   if (!pid)
    goto bad_fork_put_binfmt_module;
  }
--- ./kernel/pid.c.ve8 2007-08-06 15:39:05.000000000 +0400
```

```
+++ ./kernel/pid.c 2007-08-06 15:41:34.000000000 +0400
@@ -29,7 +29,8 @@
 #include <linux/pid_namespace.h>
 #include <linux/init_task.h>

-#define pid_hashfn(nr) hash_long((unsigned long)nr, pidhash_shift)
+#define pid_hashfn(nr, ns) \
+ hash_long((unsigned long)nr + (unsigned long)ns, pidhash_shift)
 static struct hlist_head *pid_hash;
 static int pidhash_shift;
 struct pid init_struct_pid = INIT_STRUCT_PID;
@@ -188,11 +189,13 @@ fastcall void put_pid(struct pid *pid)
  if (!pid)
   return;

- /* FIXME - this must be the namespace this pid lives in */
- ns = &init_pid_ns;
+ ns = pid->numbers[pid->level].ns;
  if ((atomic_read(&pid->count) == 1) ||
-      atomic_dec_and_test(&pid->count))
+      atomic_dec_and_test(&pid->count)) {
   kmem_cache_free(ns->pid_cachep, pid);
+  if (ns != &init_pid_ns)
+   put_pid_ns(ns);
+ }
 }
 EXPORT_SYMBOL_GPL(put_pid);

@@ -205,45 +208,61 @@ static void delayed_put_pid(struct rcu_h
 fastcall void free_pid(struct pid *pid)
 {
  /* We can be called with write_lock_irq(&tasklist_lock) held */
+ int i;
  unsigned long flags;

  spin_lock_irqsave(&pidmap_lock, flags);
  hlist_del_rcu(&pid->pid_chain);
  spin_unlock_irqrestore(&pidmap_lock, flags);

- free_pidmap(&init_pid_ns, pid->nr);
+ for (i = 0; i <= pid->level; i++)
+  free_pidmap(pid->numbers[i].ns, pid->numbers[i].nr);
+
  call_rcu(&pid->rcu, delayed_put_pid);
 }

-struct pid *alloc_pid(void)
+struct pid *alloc_pid(struct pid_namespace *ns)
```

```
 {
  struct pid *pid;
  enum pid_type type;
- int nr = -1;
- struct pid_namespace *ns;
+ int i, nr;
+ struct pid_namespace *tmp;

- ns = task_active_pid_ns(current);
  pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
  if (!pid)
   goto out;

- nr = alloc_pidmap(ns);
- if (nr < 0)
-  goto out_free;
+ tmp = ns;
+ for (i = ns->level; i >= 0; i--) {
+  nr = alloc_pidmap(tmp);
+  if (nr < 0)
+   goto out_free;
+
+  pid->numbers[i].nr = nr;
+  pid->numbers[i].ns = tmp;
+  tmp = tmp->parent;
+ }
+
+ if (ns != &init_pid_ns)
+  get_pid_ns(ns);

+ pid->level = ns->level;
+ pid->nr = pid->numbers[0].nr;
  atomic_set(&pid->count, 1);
- pid->nr = nr;
  for (type = 0; type < PIDTYPE_MAX; ++type)
   INIT_HLIST_HEAD(&pid->tasks[type]);

  spin_lock_irq(&pidmap_lock);
- hlist_add_head_rcu(&pid->pid_chain, &pid_hash[pid_hashfn(pid->nr)]);
+ hlist_add_head_rcu(&pid->pid_chain, &pid_hash[pid_hashfn(pid->nr, ns)]);
  spin_unlock_irq(&pidmap_lock);

 out:
  return pid;

 out_free:
+ for (i++; i <= ns->level; i++)
+  free_pidmap(pid->numbers[i].ns, pid->numbers[i].nr);
```

```
+
  kmem_cache_free(ns->pid_cachep, pid);
  pid = NULL;
  goto out;
@@ -255,7 +274,7 @@ struct pid * fastcall find_pid(int nr)
  struct pid *pid;

  hlist_for_each_entry_rcu(pid, elem,
-   &pid_hash[pid_hashfn(nr)], pid_chain) {
+   &pid_hash[pid_hashfn(nr, &init_pid_ns)], pid_chain) {
    if (pid->nr == nr)
     return pid;
  }
```