Subject: [PATCH 8/13] The namespace cloning
Posted by Pavel Emelianov on Thu, 24 May 2007 12:57:58 GMT
View Forum Message <> Reply to Message

This is the core of the set - the namespace cloning.

The cloning consists of two stages - creating of the new
namespace and moving a task into it. Create and move is
not good as the error path just puts the new namespaces
and thus keep the task in it.

So after the new namespace is clones task still outside it.
It is injected inside explicitly after all the operations
that might fail are finished.

Another important thing is that task must be alone in its
group and session and must not be splitted into threads.
This is because all task's pids are moved to the new ns
and if they are shared with someone else this someone may
happen to be half-inserted into the new space.

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

```
diff --git a/include/linux/pid.h b/include/linux/pid.h
index 1e0e4e3..3a30f8a 100644
--- a/include/linux/pid.h
+++ b/include/linux/pid.h
@@ -106,6 +114,8 @@ static inline pid_t pid_nr(struct pid *p

 #define pid_nr_ns(pid, ns) (ns == &init_pid_ns ? pid_nr(pid) : pid_vnr(pid))

+void move_init_to_ns(struct task_struct *tsk);
+
 #define do_each_pid_task(pid, type, task)    \
  do {        \
   struct hlist_node *pos___;    \
diff --git a/kernel/fork.c b/kernel/fork.c
index d7207a1..3ab517c 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -1659,8 +1665,11 @@ asmlinkage long sys_unshare(unsigned lon
   task_unlock(current);
  }

- if (new_nsproxy)
+ if (new_nsproxy) {
```

```
+  if (unshare_flags & CLONE_NEWPIDS)
+   move_init_to_ns(current);
   put_nsproxy(new_nsproxy);
+ }

 bad_unshare_cleanup_semundo:
 bad_unshare_cleanup_fd:
diff --git a/kernel/pid.c b/kernel/pid.c
index eb66bd2..1815af4 100644
--- a/kernel/pid.c
+++ b/kernel/pid.c
@@ -350,11 +412,99 @@ struct pid *find_get_pid(pid_t nr)
 }
 EXPORT_SYMBOL_GPL(find_get_pid);

+#ifdef CONFIG_PID_NS
+static struct pid_namespace *create_pid_namespace(void)
+{
+ struct pid_namespace *ns;
+ int i;
+
+ ns = kmalloc(sizeof(struct pid_namespace), GFP_KERNEL);
+ if (ns == NULL)
+  goto out;
+
+ ns->pidmap[0].page = kzalloc(PAGE_SIZE, GFP_KERNEL);
+ if (!ns->pidmap[0].page)
+  goto out_free;
+
+ set_bit(0, ns->pidmap[0].page);
+ atomic_set(&ns->pidmap[0].nr_free, BITS_PER_PAGE - 1);
+
+ kref_init(&ns->kref);
+ ns->last_pid = 0;
+ ns->child_reaper = NULL;
+
+ for (i = 1; i < PIDMAP_ENTRIES; i++) {
+  ns->pidmap[i].page = 0;
+  atomic_set(&ns->pidmap[i].nr_free, BITS_PER_PAGE);
+ }
+
+ return ns;
+
+out_free:
+ kfree(ns);
+out:
+ return ERR_PTR(-ENOMEM);
+}
```

```
+
+static int alone_in_pgrp(struct task_struct *tsk)
+{
+ int alone = 0;
+ struct pid *pid;
+ struct task_struct *p;
+
+ if (!thread_group_empty(tsk))
+  return 0;
+
+ read_lock(&tasklist_lock);
+ pid = tsk->pids[PIDTYPE_PGID].pid;
+ do_each_pid_task(pid, PIDTYPE_PGID, p) {
+  if (p != tsk)
+   goto out;
+ } while_each_pid_task(pid, PIDTYPE_PGID, p);
+ pid = tsk->pids[PIDTYPE_SID].pid;
+ do_each_pid_task(pid, PIDTYPE_SID, p) {
+  if (p != tsk)
+   goto out;
+ } while_each_pid_task(pid, PIDTYPE_SID, p);
+ alone = 1;
+out:
+ read_unlock(&tasklist_lock);
+ return alone;
+}
+
+static void destroy_pid_namespace(struct pid_namespace *ns)
+{
+ int i;
+
+ for (i = 0; i < PIDMAP_ENTRIES; i++)
+  kfree(ns->pidmap[i].page);
+
+ kfree(ns);
+}
+
 struct pid_namespace *copy_pid_ns(int flags, struct pid_namespace *old_ns)
 {
+ struct pid_namespace *new_ns;
+
  BUG_ON(!old_ns);
  get_pid_ns(old_ns);
- return old_ns;
+ new_ns = old_ns;
+ if (!(flags & CLONE_NEWPIDS))
+  goto out;
+
```

```
+ new_ns = ERR_PTR(-EINVAL);
+ if (old_ns != &init_pid_ns)
+  goto out_put;
+
+ new_ns = ERR_PTR(-EBUSY);
+ if (!alone_in_pgrp(current))
+  goto out_put;
+
+ new_ns = create_pid_namespace();
+out_put:
+ put_pid_ns(old_ns);
+out:
+ return new_ns;
 }

 void free_pid_ns(struct kref *kref)
@@ -377,9 +527,69 @@ void free_pid_ns(struct kref *kref)
  struct pid_namespace *ns;

  ns = container_of(kref, struct pid_namespace, kref);
- kfree(ns);
+ destroy_pid_namespace(ns);
 }

+static inline int move_pid_to_ns(struct pid *pid, struct pid_namespace *ns)
+{
+ int vnr;
+
+ vnr = alloc_pidmap(ns);
+ if (vnr < 0)
+  return -ENOMEM;
+
+ get_pid_ns(ns);
+ pid->vnr = vnr;
+ pid->ns = ns;
+ spin_lock_irq(&pidmap_lock);
+ hlist_add_head_rcu(&pid->vpid_chain,
+   &vpid_hash[vpid_hashfn(vnr, ns)]);
+ spin_unlock_irq(&pidmap_lock);
+ return 0;
+}
+
+void move_init_to_ns(struct task_struct *tsk)
+{
+ struct pid *pid;
+ struct pid_namespace *ns;
+
+ BUG_ON(tsk != current);
```

```
+ ns = tsk->nsproxy->pid_ns;
+
+ pid = tsk->pids[PIDTYPE_PID].pid;
+ BUG_ON(pid->ns != &init_pid_ns);
+ if (move_pid_to_ns(pid, ns))
+  BUG();
+ set_task_vpid(tsk, pid->vnr);
+ set_task_vtgid(tsk, pid->vnr);
+
+ pid = tsk->pids[PIDTYPE_SID].pid;
+ if (pid->ns == &init_pid_ns)
+  if (move_pid_to_ns(pid, ns))
+   BUG();
+ set_task_vsession(tsk, pid->vnr);
+
+ pid = tsk->pids[PIDTYPE_PGID].pid;
+ if (pid->ns == &init_pid_ns)
+  if (move_pid_to_ns(pid, ns))
+   BUG();
+ set_task_vpgrp(tsk, pid->vnr);
+
+ ns->child_reaper = tsk;
+}
+#else
+struct pid_namespace *copy_pid_ns(int flags, struct pid_namespace *old_ns)
+{
+ BUG_ON(flags & CLONE_NEWPIDS);
+ return old_ns;
+}
+
+void move_init_to_ns(struct task_struct *tsk)
+{
+ BUG();
+}
+#endif
+
 /*
  * The pid hash table is scaled according to the amount of memory in the
  * machine.  From a minimum of 16 slots up to 4096 slots at one gigabyte or
```