Subject: [PATCH 15/28] The namespace cloning
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:13:17 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>

```
 include/linux/pid_namespace.h |    2
 include/linux/sched.h         |    1
 kernel/fork.c                 |   15 +++-
 kernel/nsproxy.c              |    6 +
 kernel/pid.c                  |  152 ++++++++++++++++++++++++++++++++++++++++++-
 5 files changed, 172 insertions(+), 4 deletions(-)

--- ./include/linux/pid_namespace.h.clonepidns 2007-06-15 15:13:07.000000000 +0400
+++ ./include/linux/pid_namespace.h 2007-06-15 15:14:33.000000000 +0400
@@ -50,4 +50,6 @@ static inline struct task_struct *child_
  return tsk->nsproxy->pid_ns->child_reaper;
 }

+int move_init_to_ns(struct task_struct *tsk, struct nsproxy *nsp);
+
 #endif /* _LINUX_PID_NS_H */
--- ./include/linux/sched.h.clonepidns 2007-06-15 15:00:44.000000000 +0400
+++ ./include/linux/sched.h 2007-06-15 15:14:33.000000000 +0400
@@ -25,6 +25,7 @@
 #define CLONE_STOPPED  0x02000000 /* Start in stopped state */
 #define CLONE_NEWUTS  0x04000000 /* New utsname group? */
 #define CLONE_NEWIPC  0x08000000 /* New ipcs */
+#define CLONE_NEWPIDS  0x10000000 /* New pids */

 /*
  * Scheduling policies
```

```
--- ./kernel/fork.c.clonepidns 2007-06-15 15:02:29.000000000 +0400
+++ ./kernel/fork.c 2007-06-15 15:18:15.000000000 +0400
@@ -1623,7 +1623,7 @@ asmlinkage long sys_unshare(unsigned lon
  err = -EINVAL;
  if (unshare_flags & ~(CLONE_THREAD|CLONE_FS|CLONE_NEWNS|CLONE_SIGHAND|
     CLONE_VM|CLONE_FILES|CLONE_SYSVSEM|
-    CLONE_NEWUTS|CLONE_NEWIPC))
+    CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWPIDS))
   goto bad_unshare_out;

  if ((err = unshare_thread(unshare_flags)))
@@ -1642,6 +1642,18 @@ asmlinkage long sys_unshare(unsigned lon
    new_fs)))
   goto bad_unshare_cleanup_semundo;

+ if (new_nsproxy && (unshare_flags & CLONE_NEWPIDS))
+  /*
+   * when we created new pid namespace and failed with something
+   * later we cannot roll back the pid ns creation with simple
+   * put_pid_ns as the task will stay in this namespace holding
+   * it. there are two solutions - pull the task out of this ns
+   * on the error path or push the task into it on the sucess one
+   * I use the second way.
+   */
+  if (move_init_to_ns(current, new_nsproxy))
+   goto bad_move_to_ns;
+
  if (new_fs ||  new_mm || new_fd || new_ulist || new_nsproxy) {

   task_lock(current);
@@ -1676,6 +1688,7 @@ asmlinkage long sys_unshare(unsigned lon
   task_unlock(current);
  }

+bad_move_to_ns:
  if (new_nsproxy)
   put_nsproxy(new_nsproxy);

--- ./kernel/nsproxy.c.clonepidns 2007-06-15 15:00:32.000000000 +0400
+++ ./kernel/nsproxy.c 2007-06-15 15:14:33.000000000 +0400
@@ -110,6 +110,9 @@ int copy_namespaces(int flags, struct ta

  get_nsproxy(old_ns);

+ if (flags & CLONE_NEWPIDS)
+  return -EINVAL;
+
  if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC)))
```

```
   return 0;

@@ -154,7 +157,8 @@ int unshare_nsproxy_namespaces(unsigned
  struct nsproxy *old_ns = current->nsproxy;
  int err = 0;

- if (!(unshare_flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC)))
+ if (!(unshare_flags & (CLONE_NEWNS | CLONE_NEWUTS |
+     CLONE_NEWIPC | CLONE_NEWPIDS)))
   return 0;

 #ifndef CONFIG_IPC_NS
--- ./kernel/pid.c.clonepidns 2007-06-15 15:13:07.000000000 +0400
+++ ./kernel/pid.c 2007-06-15 15:14:33.000000000 +0400
@@ -28,6 +28,7 @@
 #include <linux/hash.h>
 #include <linux/pid_namespace.h>
 #include <linux/init_task.h>
+#include <linux/proc_fs.h>

 #define pid_hashfn(nr) hash_long((unsigned long)nr, pidhash_shift)
 static struct hlist_head *pid_hash;
@@ -401,11 +402,102 @@ struct pid *find_ge_pid(int nr, struct p
 }
 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;
+
+ if (pid_ns_prepare_proc(ns))
+  goto out_free_map;
+
+ 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_map:
+ kfree(ns->pidmap[0].page);
+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;
+
+ pid_ns_release_proc(ns);
```

```
+
+ 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(-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)
@@ -413,8 +505,64 @@ 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);
+}
+
+int move_init_to_ns(struct task_struct *tsk, struct nsproxy *nsp)
+{
+ int err;
+ struct pid *pid;
+ struct pid_namespace *ns;
+
+ BUG_ON(tsk != current);
+ ns = nsp->pid_ns;
+
+ pid = task_pid(tsk);
```

```
+ err = move_pid_to_ns(pid, ns);
+ if (err < 0)
+  goto out_pid;
+
+ set_task_vpid(tsk, pid_vnr(pid));
+ set_task_vtgid(tsk, pid_vnr(pid));
+
+ pid = task_session(tsk);
+ err = move_pid_to_ns(pid, ns);
+ if (err < 0)
+  goto out_sid;
+
+ set_task_vsession(tsk, pid_vnr(pid));
+
+ pid = task_pgrp(tsk);
+ err = move_pid_to_ns(pid, ns);
+ if (err < 0)
+  goto out_pgid;
+
+ set_task_vpgrp(tsk, pid_vnr(pid));
+
+ ns->child_reaper = tsk;
+ return 0;
+
+out_pgid:
+ del_pid_from_ns(task_session(tsk), ns);
+out_sid:
+ del_pid_from_ns(task_pid(tsk), ns);
+out_pid:
+ return err;
 }
+#else
+struct pid_namespace *copy_pid_ns(int flags, struct pid_namespace *old_ns)
+{
+ if (flags & CLONE_NEWPIDS)
+  old_ns = ERR_PTR(-EINVAL);
+
+ return old_ns;
+}
+
+int move_init_to_ns(struct task_struct *tsk, struct nsproxy *nsp)
+{
+ BUG();
+}
+#endif

 /*
  * The pid hash table is scaled according to the amount of memory in the
```

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