Subject: [PATCH 14/16] Make pid namespaces clonnable
Posted by Pavel Emelianov on Fri, 06 Jul 2007 08:11:18 GMT
View Forum Message <> Reply to Message

Just add the support for cloning pid namespaces.

Note that the namespace is destroyed via schedule_work(). This is
done so, since the namespace will put the proc_mnt and thus may
fall asleep.

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

```
 include/linux/pid_namespace.h |    5 +-
 include/linux/sched.h         |    1
 kernel/fork.c                 |   26 ++++++---
 kernel/nsproxy.c              |    2
 kernel/pid.c                  |   96 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++--
 5 files changed, 118 insertions(+), 12 deletions(-)

--- ./include/linux/pid_namespace.h.ve13 2007-07-06 11:05:05.000000000 +0400
+++ ./include/linux/pid_namespace.h 2007-07-06 11:05:05.000000000 +0400
@@ -15,7 +15,10 @@ struct pidmap {
 #define PIDMAP_ENTRIES         ((PID_MAX_LIMIT + 8*PAGE_SIZE - 1)/PAGE_SIZE/8)

 struct pid_namespace {
- struct kref kref;
+ union {
+  struct kref kref;
+  struct work_struct free_work;
+ };
   struct pidmap pidmap[PIDMAP_ENTRIES];
   int last_pid;
   int level;
--- ./include/linux/sched.h.ve13 2007-07-06 11:05:05.000000000 +0400
+++ ./include/linux/sched.h 2007-07-06 11:05:23.000000000 +0400
@@ -26,6 +26,7 @@
 #define CLONE_NEWUTS  0x04000000 /* New utsname group? */
 #define CLONE_NEWIPC  0x08000000 /* New ipcs */
 #define CLONE_NEWUSER  0x10000000 /* New user namespace */
+#define CLONE_NEWPIDS  0x20000000 /* New pids */

 /*
   * Scheduling policies
--- ./kernel/fork.c.ve13 2007-07-06 11:05:05.000000000 +0400
+++ ./kernel/fork.c 2007-07-06 11:05:05.000000000 +0400
@@ -1267,11 +1267,22 @@ static struct task_struct *copy_process(
```

```
    __ptrace_link(p, current->parent);

   if (thread_group_leader(p)) {
-    p->signal->tty = current->signal->tty;
-    p->signal->pgrp = task_pgrp_nr(current);
-    set_task_session(p, task_session_nr(current));
-    attach_pid(p, PIDTYPE_PGID, task_pgrp(current));
-    attach_pid(p, PIDTYPE_SID, task_session(current));
+    if (clone_flags & CLONE_NEWPIDS) {
+     p->nsproxy->pid_ns->child_reaper = p;
+     p->signal->tty = NULL;
+     p->signal->pgrp = p->pid;
+     set_task_session(p, p->pid);
+     attach_pid(p, PIDTYPE_PGID, pid);
+     attach_pid(p, PIDTYPE_SID, pid);
+    } else {
+     p->signal->tty = current->signal->tty;
+     p->signal->pgrp = task_pgrp_nr(current);
+     set_task_session(p, task_session_nr(current));
+     attach_pid(p, PIDTYPE_PGID,
+       task_pgrp(current));
+     attach_pid(p, PIDTYPE_SID,
+       task_session(current));
+    }

    list_add_tail_rcu(&p->tasks, &init_task.tasks);
    __get_cpu_var(process_counts)++;
@@ -1423,7 +1434,10 @@ long do_fork(unsigned long clone_flags,
   else
    p->state = TASK_STOPPED;

-  nr = pid_vnr(task_pid(p));
+  nr = (clone_flags & CLONE_NEWPIDS) ?
+    pid_nr_ns(task_pid(p), current->nsproxy->pid_ns) :
+    pid_vnr(task_pid(p));
+
   if (unlikely (trace)) {
    current->ptrace_message = nr;
    ptrace_notify ((trace << 8) | SIGTRAP);
--- ./kernel/nsproxy.c.ve13 2007-07-06 10:58:57.000000000 +0400
+++ ./kernel/nsproxy.c 2007-07-06 11:05:39.000000000 +0400
@@ -184,7 +184,7 @@ int unshare_nsproxy_namespaces(unsigned
  int err = 0;

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

```
  if (!capable(CAP_SYS_ADMIN))
--- ./kernel/pid.c.ve13 2007-07-06 11:05:05.000000000 +0400
+++ ./kernel/pid.c 2007-07-06 11:06:47.000000000 +0400
@@ -62,8 +62,10 @@ static inline int mk_pid(struct pid_name
  * the scheme scales to up to 4 million PIDs, runtime.
  */
 struct pid_namespace init_pid_ns = {
- .kref = {
-  .refcount       = ATOMIC_INIT(2),
+ {
+  .kref = {
+   .refcount       = ATOMIC_INIT(2),
+  },
  },
  .pidmap = {
   [ 0 ... PIDMAP_ENTRIES-1] = { ATOMIC_INIT(BITS_PER_PAGE), NULL }
@@ -457,11 +459,96 @@ out:
  return cachep;
 }

+static struct pid_namespace *create_pid_namespace(int level)
+{
+ 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;
+
+ ns->pid_cachep = create_pid_cachep(level);
+ if (ns->pid_cachep == NULL)
+  goto out_free_map;
+
+ kref_init(&ns->kref);
+ ns->last_pid = 0;
+ ns->child_reaper = NULL;
+ ns->level = level;
+
+ if (pid_ns_prepare_proc(ns))
+  goto out_free_cachep;
+
+ set_bit(0, ns->pidmap[0].page);
+ atomic_set(&ns->pidmap[0].nr_free, BITS_PER_PAGE - 1);
```

```
+ get_pid_ns(ns);
+
+ 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_cachep:
+out_free_map:
+ kfree(ns->pidmap[0].page);
+out_free:
+ kfree(ns);
+out:
+ return ERR_PTR(-ENOMEM);
+}
+
+static void destroy_pid_namespace(struct pid_namespace *ns)
+{
+ int i;
+
+ synchronize_rcu();
+ pid_ns_release_proc(ns);
+ atomic_inc(&ns->pidmap[0].nr_free);
+ for (i = 0; i < PIDMAP_ENTRIES; i++)
+  kfree(ns->pidmap[i].page);
+ kfree(ns);
+}
+
 struct pid_namespace *copy_pid_ns(unsigned long 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 (flags & CLONE_THREAD)
+  goto out_put;
+
+ new_ns = create_pid_namespace(old_ns->level + 1);
+ if (new_ns != NULL)
+  new_ns->parent = get_pid_ns(old_ns);
```

```
+out_put:
+ put_pid_ns(old_ns);
+out:
+ return new_ns;
+}
+
+static void do_free_pid_ns(struct work_struct *w)
+{
+ struct pid_namespace *ns, *parent;
+
+ ns = container_of(w, struct pid_namespace, free_work);
+ parent = ns->parent;
+ destroy_pid_namespace(ns);
+
+ if (parent != NULL)
+  put_pid_ns(parent);
 }

 void free_pid_ns(struct kref *kref)
@@ -469,7 +556,8 @@ void free_pid_ns(struct kref *kref)
  struct pid_namespace *ns;

  ns = container_of(kref, struct pid_namespace, kref);
- kfree(ns);
+ INIT_WORK(&ns->free_work, do_free_pid_ns);
+ schedule_work(&ns->free_work);
 }

 /*
```

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