Subject: [PATCH 13/15] Clone the pid namespace
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:56:50 GMT
View Forum Message <> Reply to Message

When clone() is invoked with CLONE_NEWPID, create a new pid namespace
Since the active pid namespace is special and expected to be the first
entry in pid->upid_list, preserve the order of pid namespaces.

Pid namespaces can be nested and the nesting depth is unlimited.
When a process clones its pid namespace, we create additional pid caches
as necessary and use the pid cache to allocate 'struct pids' for that depth.

TODO:
 One of the reasons the free_work() was introduced was to cleanup
 the proc in non-atomic context, but since proc is now released
 from proc_flush_task() this looks to be unneeded, but I have to
 recheck this.

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

---

 include/linux/pid_namespace.h | 7 ++
 include/linux/sched.h | 1
 kernel/nsproxy.c | 3 -
 kernel/pid.c | 102 +++++++++++++++++++++++++++++++++++-----
 4 files changed, 101 insertions(+), 12 deletions(-)

diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/pid_namespace.h
linux-2.6.23-rc1-mm1-7/include/linux/pid_namespace.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/pid_namespace.h 2007-07-26 16:34:45.000000000
+0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/pid_namespace.h 2007-07-26 16:36:36.000000000
+0400
@@ -16,11 +15,16 @@ 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;
  struct task_struct *child_reaper;
  struct kmem_cache *pid_cachep;
+ int level;

```
+ struct pid_namespace *parent;
 #ifdef CONFIG_PROC_FS
  struct vfsmount *proc_mnt;
 #endif
diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/sched.h
linux-2.6.23-rc1-mm1-7/include/linux/sched.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/sched.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/sched.h 2007-07-26 16:36:37.000000000 +0400
@@ -27,6 +27,7 @@
 #define CLONE_NEWUTS  0x04000000 /* New utsname group? */
 #define CLONE_NEWIPC  0x08000000 /* New ipcs */
 #define CLONE_NEWUSER  0x10000000 /* New user namespace */
+#define CLONE_NEWPID  0x20000000 /* New pids */

 /*
  * Scheduling policies
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/nsproxy.c linux-2.6.23-rc1-mm1-7/kernel/nsproxy.c
--- linux-2.6.23-rc1-mm1.orig/kernel/nsproxy.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/nsproxy.c 2007-07-26 16:36:36.000000000 +0400
@@ -132,7 +132,8 @@ int copy_namespaces(unsigned long flags,

  get_nsproxy(old_ns);

- if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC | CLONE_NEWUSER)))
+ if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
+     CLONE_NEWUSER | CLONE_NEWPID)))
   return 0;

  if (!capable(CAP_SYS_ADMIN)) {
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/pid.c linux-2.6.23-rc1-mm1-7/kernel/pid.c
--- linux-2.6.23-rc1-mm1.orig/kernel/pid.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/pid.c 2007-07-26 16:36:37.000000000 +0400
@@ -60,14 +62,17 @@ 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 }
  },
  .last_pid = 0,
- .child_reaper = &init_task
```

```
+ .level = 0,
+ .child_reaper = &init_task,
 };
 EXPORT_SYMBOL(init_pid_ns);

@@ -409,8 +501,8 @@ static struct kmem_cache *create_pid_cac

  snprintf(pcache->name, sizeof(pcache->name), "pid_%d", nr_ids);
  cachep = kmem_cache_create(pcache->name,
-   /* FIXME add numerical ids here */
-   sizeof(struct pid), 0, SLAB_HWCACHE_ALIGN, NULL);
+   sizeof(struct pid) + (nr_ids - 1) * sizeof(struct upid),
+   0, SLAB_HWCACHE_ALIGN, NULL);
  if (cachep == NULL)
   goto err_cachep;

@@ -428,11 +520,89 @@ err_alloc:
  return NULL;
 }

-struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *old_ns)
+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 + 1);
+ if (ns->pid_cachep == NULL)
+  goto out_free_map;
+
+ kref_init(&ns->kref);
+ ns->last_pid = 0;
+ ns->child_reaper = NULL;
+ ns->level = level;
+
+ 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_map:
+ kfree(ns->pidmap[0].page);
+out_free:
+ kfree(ns);
+out:
+ return ERR_PTR(-ENOMEM);
+}
+
+static void destroy_pid_namespace(struct pid_namespace *ns)
 {
- BUG_ON(!old_ns);
- get_pid_ns(old_ns);
- return old_ns;
+ int i;
+
+ 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);
+ new_ns = get_pid_ns(old_ns);
+ if (!(flags & CLONE_NEWPID))
+  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)
@@ -440,7 +648,8 @@ 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);
 }

 /*
```