
Subject: [PATCH 14/20] Allow cloning of new namespace
Posted by [Pavel Emelianov](#) on Tue, 07 Aug 2007 09:30:02 GMT
[View Forum Message](#) <> [Reply to Message](#)

When clone() is invoked with CLONE_NEWPID, create a new pid namespace and then create a new struct pid for the new process. Allocate pid_t's for the new process in the new pid namespace and all ancestor pid namespaces. Make the newly cloned process the session and process group leader.

Since the active pid namespace is special and expected to be the first entry in pid->upid_list, preserve the order of pid namespaces.

The size of 'struct pid' is dependent on the the number of pid namespaces the process exists in, so we use multiple pid-caches'. Only one pid cache is created during system startup and this used by processes that exist only in init_pid_ns.

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.

Note, that with this patch the newly created namespace won't work, since the rest of the kernel still uses global pids, but this is to be fixed soon. Init pid namespace still works.

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

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>

```
include/linux/sched.h | 1
kernel/fork.c          | 48 ++++++-----
kernel/nsproxy.c       | 3 +
kernel/pid.c           | 91 ++++++-----
4 files changed, 118 insertions(+), 25 deletions(-)
```

--- ./include/linux/sched.h.ve14 2007-08-06 15:00:09.000000000 +0400

+++ ./include/linux/sched.h 2007-08-06 15:00:09.000000000 +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

--- ./kernel/nsproxy.c.ve14 2007-08-06 15:00:05.000000000 +0400

+++ ./kernel/nsproxy.c 2007-08-06 15:00:09.000000000 +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)) {
--- ./kernel/pid.c.ve14 2007-08-06 15:00:09.000000000 +0400
+++ ./kernel/pid.c 2007-08-06 15:03:29.000000000 +0400
@@ -18,6 +18,11 @@
 * allocation scenario when all but one out of 1 million PIDs possible are
 * allocated already: the scanning of 32 list entries and at most PAGE_SIZE
 * bytes. The typical fastpath is a single successful setbit. Freeing is O(1).
+ *
+ * Pid namespaces:
+ * (C) 2007 Pavel Emelyanov <xemul@openvz.org>, OpenVZ, SWsoft Inc.
+ * (C) 2007 Sukadev Bhattiprolu <sukadev@us.ibm.com>, IBM
+ *
*/

#include <linux/mm.h>
@@ -453,8 +453,8 @@ 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;

@@ -472,19 +472,89 @@ 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)
{
-   BUG_ON(!old_ns);
-   get_pid_ns(old_ns);
-   return old_ns;
+   BUG_ON(!old_ns);
+   get_pid_ns(old_ns);
+   return old_ns;
+   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_cache = create_pid_cache(level + 1);
+ if (ns->pid_cache == 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);
+
+ 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)
+{
+ 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;
}

void free_pid_ns(struct kref *kref)
{
- struct pid_namespace *ns;
+ struct pid_namespace *ns, *parent;

    ns = container_of(kref, struct pid_namespace, kref);
- kfree(ns);
+
+ parent = ns->parent;
+ destroy_pid_namespace(ns);
+
+ if (parent != NULL)
+ put_pid_ns(parent);
}

/*
--- ./kernel/fork.c.ve12 2007-08-06 12:44:54.000000000 +0400
+++ ./kernel/fork.c 2007-08-06 12:44:57.000000000 +0400
@@ -961,7 +961,6 @@ static struct task_struct *copy_process(
    unsigned long stack_start,
    struct pt_regs *regs,
    unsigned long stack_size,
-   int __user *parent_tidptr,
    int __user *child_tidptr,
    struct pid *pid)
{
@@ -1266,11 +1266,22 @@ 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_NEWPID) {
+   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)++;
@@ -1283,13 +1294,6 @@ static struct task_struct *copy_process(
    spin_unlock(&current->sigand->siglock);
    write_unlock_irq(&tasklist_lock);

- /*
-  * Now that we know the fork has succeeded, record the new
-  * TID. It's too late to back out if this fails.
-  */
- if (clone_flags & CLONE_PARENT_SETTID)
-   put_user(p->pid, parent_tidptr);
-
    proc_fork_connector(p);
    container_post_fork(p);
    return p;
@@ -1350,7 +1354,7 @@ struct task_struct * __cpuinit fork_idle
    struct task_struct *task;
    struct pt_regs regs;

- task = copy_process(CLONE_VM, 0, idle_regs(&regs), 0, NULL, NULL,
+ task = copy_process(CLONE_VM, 0, idle_regs(&regs), 0, NULL,
    &init_struct_pid);
    if (!IS_ERR(task))
        init_idle(task, cpu);

```

```

@@ -1398,7 +1402,7 @@ long do_fork(unsigned long clone_flags,
}

p = copy_process(clone_flags, stack_start, regs, stack_size,
- parent_tidptr, child_tidptr, NULL);
+ child_tidptr, NULL);
/*
 * Do this prior waking up the new thread - the thread pointer
 * might get invalid after that point, if the thread exits quickly.
@@ -1406,7 +1410,20 @@ long do_fork(unsigned long clone_flags,
if (!IS_ERR(p)) {
    struct completion vfork;

- nr = pid_nr(task_pid(p));
+ /*
+  * this is enough to call pid_nr_ns here, but this if
+  * improves optimisation of regular fork()
+  */
+ nr = (clone_flags & CLONE_NEWPID) ?
+ task_pid_nr_ns(p, current->nsproxy->pid_ns) :
+ task_pid_vnr(p);
+
+ /*
+  * Now that we know the fork has succeeded, record the new
+  * TID. It's too late to back out if this fails.
+  */
+ if (clone_flags & CLONE_PARENT_SETTID)
+ put_user(nr, parent_tidptr);

    if (clone_flags & CLONE_VFORK) {
        p->vfork_done = &vfork;

```
