Subject: Re: [PATCH] allow a task to join a pid namespace
Posted by ebiederm on Wed, 06 Jun 2012 18:29:16 GMT
View Forum Message <> Reply to Message

Glauber Costa <glommer@parallels.com> writes:
>
> Please let me know what you think of the attached patch. It is ontop of Eric's
> tree that you pointed me to, but I am not really using any of its functionality,
> so this would be equally doable in current mainline kernel - but I wanted to
> make sure it integrates well with what Eric is doing as well.

The trees I have out there for pid namespace work are for historical
reference at this point.

> It is a bit wasteful space-wise, but this approach pretty much guarantees we
> don't need to update pointers anywhere - therefore, totally
> lock-free. pid->level is only updated after switch_task_namespaces(), so every
> call before that will see correct information up to the previous
> level.

This solves none of the pid namespace life cycle problems and the issues
you solve by extending a pid can probably be solved better by allocating
a new pid and setting it's low pids the same as the original pid and
then atomically swapping them.

So I do not find the patch below at all interesting.

Eric

> From 1192e0ff3c854f4690d44fadccbc575fff653578 Mon Sep 17 00:00:00 2001
> From: Glauber Costa <glommer@parallels.com>
> Date: Tue, 5 Jun 2012 15:23:12 +0400
> Subject: [PATCH] pid_ns: expand the current pid namespace to a new namespace
>
> install pid_ns doesn't go as far as needed. Things like
> ns_of_pid() will rely on the current level to get to the
> right task. So when attaching a task to a new pid namespace,
> we need to make sure the upid vector grows as needed, and the
> level updated accordingly
>
> To do that, this patch leaves room for one more upid at the end
> of the structure at all times. It also stores the original level so
> we know afterwards which cache to free from.
>
> Although this is, of course, a bit wasteful, it has the advantage
> that we never need to touch the existing struct pid, nor the existing
> upid vectors. That would be racy in nature, and may require us to pick

> locks here and there - which seems to me like a big no.
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> ---
> include/linux/pid.h | 3 +++
> kernel/nsproxy.c | 2 ++
> kernel/pid.c | 54 ++-
> kernel/pid_namespace.c | 30 ++++++++++++++++++++++++++--
> 4 files changed, 86 insertions(+), 3 deletions(-)
>
> diff --git a/include/linux/pid.h b/include/linux/pid.h
> index b152d44..2f01763 100644
> --- a/include/linux/pid.h
> +++ b/include/linux/pid.h
> @@ -58,6 +58,7 @@ struct pid
> {
>   atomic_t count;
>   unsigned int level;
> + unsigned int orig_level;
>   /* lists of tasks that use this pid */
>   struct hlist_head tasks[PIDTYPE_MAX];
>   struct rcu_head rcu;
> @@ -121,6 +122,8 @@ int next_pidmap(struct pid_namespace *pid_ns, unsigned int last);
>
> extern struct pid *alloc_pid(struct pid_namespace *ns);
> extern void free_pid(struct pid *pid);
> +extern int expand_pid(struct pid *pid, struct pid_namespace *ns);
> +extern void update_expanded_pid(struct task_struct *task);
>
> /*
>   * ns_of_pid() returns the pid namespace in which the specified pid was
> diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
> index b956079..9851f11 100644
> --- a/kernel/nsproxy.c
> +++ b/kernel/nsproxy.c
> @@ -267,6 +267,8 @@ SYSCALL_DEFINE2(setns, int, fd, int, nstype)
>   goto out;
>   }
>   switch_task_namespaces(tsk, new_nsproxy);
> + if (nstype == CLONE_NEWPID)
> + update_expanded_pid(current);
> out:
>   fput(file);
>   return err;
> diff --git a/kernel/pid.c b/kernel/pid.c
> index 8c51c26..f2b1bd7 100644
> --- a/kernel/pid.c
> +++ b/kernel/pid.c

```
> @@ -247,7 +247,7 @@ void put_pid(struct pid *pid)
>   if (!pid)
>    return;
>
> - ns = pid->numbers[pid->level].ns;
> + ns = pid->numbers[pid->orig_level].ns;
>   if ((atomic_read(&pid->count) == 1) ||
>        atomic_dec_and_test(&pid->count)) {
>    kmem_cache_free(ns->pid_cachep, pid);
> @@ -306,6 +306,7 @@ struct pid *alloc_pid(struct pid_namespace *ns)
>
>   get_pid_ns(ns);
>   pid->level = ns->level;
> + pid->orig_level = ns->level;
>   atomic_set(&pid->count, 1);
>   for (type = 0; type < PIDTYPE_MAX; ++type)
>    INIT_HLIST_HEAD(&pid->tasks[type]);
> @@ -329,6 +330,57 @@ out_free:
>   goto out;
>  }
>
> +int expand_pid(struct pid *pid, struct pid_namespace *ns)
> +{
> + int i, nr;
> + struct pid_namespace *tmp;
> + struct upid *upid;
> +
> + if  ((ns->level - pid->orig_level) > 1)
> +  return -EINVAL;
> +
> + if (ns->parent != pid->numbers[pid->orig_level].ns)
> +  return -EINVAL;
> +
> + tmp = ns;
> + for (i = ns->level; i > pid->level; i--) {
> +  if (ns->dead)
> +   goto out_free;
> +  nr = alloc_pidmap(tmp);
> +  if (nr < 0)
> +   goto out_free;
> +
> +  pid->numbers[i].nr = nr;
> +  pid->numbers[i].ns = tmp;
> +  tmp = tmp->parent;
> + }
> +
> + upid = pid->numbers + ns->level;
> + spin_lock_irq(&pidmap_lock);
```

```
> + for (i = ns->level; i > pid->level; i--) {
> +  upid = &pid->numbers[i];
> +  hlist_add_head_rcu(&upid->pid_chain,
> +    &pid_hash[pid_hashfn(upid->nr, upid->ns)]);
> + }
> + spin_unlock_irq(&pidmap_lock);
> +
> + return 0;
> +
> +out_free:
> + while (++i <= ns->level)
> +  free_pidmap(pid->numbers + i);
> +
> + return -ENOMEM;
> +}
> +
> +void update_expanded_pid(struct task_struct *task)
> +{
> + struct pid *pid;
> + pid = get_task_pid(task, PIDTYPE_PID);
> +
> + pid->level++;
> +}
> +
>  struct pid *find_pid_ns(int nr, struct pid_namespace *ns)
>  {
>   struct hlist_node *elem;
> diff --git a/kernel/pid_namespace.c b/kernel/pid_namespace.c
> index 79bf459..0f3a521 100644
> --- a/kernel/pid_namespace.c
> +++ b/kernel/pid_namespace.c
> @@ -49,8 +49,12 @@ static struct kmem_cache *create_pid_cachep(int nr_ids)
>    goto err_alloc;
>
>   snprintf(pcache->name, sizeof(pcache->name), "pid_%d", nr_ids);
> + /*
> +  * nr_ids - 1 would provide room for upids up to the right level.
> +  * We leave room for one in the end for usage with setns.
> +  */
>   cachep = kmem_cache_create(pcache->name,
> -   sizeof(struct pid) + (nr_ids - 1) * sizeof(struct upid),
> +   sizeof(struct pid) + (nr_ids) * sizeof(struct upid),
>     0, SLAB_HWCACHE_ALIGN, NULL);
>   if (cachep == NULL)
>    goto err_cachep;
> @@ -244,8 +248,30 @@ static void pidns_put(void *ns)
>   put_pid_ns(ns);
>  }
```

```
>
> -static int pidns_install(struct nsproxy *nsproxy, void *ns)
> +static int pidns_install(struct nsproxy *nsproxy, void *_ns)
>  {
> + int ret = 0;
> + struct pid *pid;
> + struct pid_namespace *ns = _ns;
> +
> + rcu_read_lock();
> + pid = task_pid(current);
> + rcu_read_unlock();
> +
> + if (is_container_init(current) || (get_nr_threads(current) > 1))
> +  return -EINVAL;
> +
> + read_lock(&tasklist_lock);
> + if ((task_pgrp(current) != pid) || task_session(current) != pid)
> +  ret = -EPERM;
> + read_unlock(&tasklist_lock);
> + if (ret)
> +  return ret;
> +
> + ret = expand_pid(pid, ns);
> + if (ret)
> +  return ret;
> +
>   put_pid_ns(nsproxy->pid_ns);
>   nsproxy->pid_ns = get_pid_ns(ns);
>   return 0;
```