Subject: Re: [RFC][PATCH 3/5] Use pid namespace from struct pid_nrs list
Posted by ebiederm on Sun, 11 Mar 2007 12:48:04 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com writes:

> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> Subject: [RFC][PATCH 3/5] Use pid namespace from struct pid_nrs list
>
> Stop using task->nsproxy->pid_ns. Use pid_namespace from pid->pid_nrs
> list instead.
>
> To simplify error handling, this patch moves processing of CLONE_NEWPID
> flag, currently in copy_namespaces()/copy_process(), to alloc_pid() which
> is where the process association with a pid namespace is established.
>
> i.e when cloning a new pid namespace, alloc_pid() allocates a new pid_nr
> for both the parent and child namespaces.

This patch seems to do a bit much, it is hard to follow what changes you
are making.

It probably makes sense to modify things so alloc_pid can do everything
it needs to.

It looks like we can safely move alloc_pid into copy_process and
just dig out the pid number and place it in nr if copy_process succeeds.

Which should allow the special case for setting the child reaper to go
away, because we can allocate the task_struct before allocating the struct
pid.

> Changelog:
> - Move definition of set_pid_ns_child_reaper() into the previous
>   helper function patch.
> - Minor changes to accomodate changes in underlying patches.
> - Fix compile warnings about parent_pid_ns (Cedric Le Goater)
> - [Badari Pulavarty's comments]: No need to allocate nsproxy when
>   only CLONE_NEWPID is set. And attach_pid_nr() only needs one
>   parameter.
> - Add privilege check for clone(CLONE_NEWPID).
>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> Cc: Cedric Le Goater <clg@fr.ibm.com>
> Cc: Dave Hansen <haveblue@us.ibm.com>
> Cc: Serge Hallyn <serue@us.ibm.com>
> Cc: containers@lists.osdl.org

```
>
> ---
>  include/linux/pid.h           |    2 -
>  include/linux/pid_namespace.h |   11 ++-------
>  kernel/fork.c                 |   12 ++++++---
>  kernel/nsproxy.c              |   11 ---------
>  kernel/pid.c | 51 ++++++++++++++++++++++++++++++------------
>  5 files changed, 50 insertions(+), 37 deletions(-)
>
> Index: lx26-20-mm2b/include/linux/pid.h
> ===================================================================
> --- lx26-20-mm2b.orig/include/linux/pid.h 2007-03-09 19:00:11.000000000 -0800
> +++ lx26-20-mm2b/include/linux/pid.h 2007-03-09 19:01:09.000000000 -0800
> @@ -119,7 +119,7 @@ extern struct pid *find_ge_pid(int nr);
>  extern int attach_pid_nr(struct pid *pid, struct pid_nr *pid_nr);
>  extern void free_pid_nr(struct pid_nr *pid_nr);
>  extern struct pid_nr *alloc_pid_nr(struct pid_namespace *pid_ns);
> -extern struct pid *alloc_pid(void);
> +extern struct pid *alloc_pid(int clone_flags);
>  extern void FASTCALL(free_pid(struct pid *pid));
>  extern pid_t pid_nr(struct pid *pid);
>
> Index: lx26-20-mm2b/kernel/fork.c
> ===================================================================
> --- lx26-20-mm2b.orig/kernel/fork.c 2007-03-09 19:00:42.000000000 -0800
> +++ lx26-20-mm2b/kernel/fork.c 2007-03-09 19:01:09.000000000 -0800
> @@ -1144,6 +1144,8 @@ static struct task_struct *copy_process(
>   if (clone_flags & CLONE_THREAD)
>    p->tgid = current->tgid;
>
> + if ((retval = priv_check_pid_ns(clone_flags)))
> +  goto bad_fork_cleanup_policy;
>   if ((retval = security_task_alloc(p)))
>    goto bad_fork_cleanup_policy;
>   if ((retval = audit_alloc(p)))
> @@ -1169,6 +1171,9 @@ static struct task_struct *copy_process(
>   if (retval)
>    goto bad_fork_cleanup_namespaces;
>
> + /* We are now ready to set child reaper if we cloned pid ns */
> + set_pid_ns_child_reaper(clone_flags, pid, p);
> +
>   p->set_child_tid = (clone_flags & CLONE_CHILD_SETTID) ? child_tidptr :
> NULL;
>   /*
>    * Clear TID on mm_release()?
> @@ -1384,7 +1389,7 @@ long do_fork(unsigned long clone_flags,
>        int __user *child_tidptr)
```

```
>  {
>   struct task_struct *p;
> - struct pid *pid = alloc_pid();
> + struct pid *pid = alloc_pid(clone_flags);
>   long nr;
>
>   if (!pid)
> @@ -1671,7 +1676,7 @@ asmlinkage long sys_unshare(unsigned lon
>   if ((err = unshare_pid_ns(unshare_flags, &new_pid_nr)))
>    goto bad_unshare_cleanup_ipc;
>
> - if (new_ns || new_uts || new_ipc || new_pid_nr) {
> + if (new_ns || new_uts || new_ipc) {
>    old_nsproxy = current->nsproxy;
>    new_nsproxy = dup_nsproxy(old_nsproxy);
>    if (!new_nsproxy) {
> @@ -1730,8 +1735,7 @@ asmlinkage long sys_unshare(unsigned lon
>    }
>
>    if (new_pid_nr) {
> -   pid = task_pid_ns(current);
> -   set_task_pid_ns(current, new_pid_nr->pid_ns);
> +   attach_pid_nr(task_pid(current), new_pid_nr);
>     new_pid_nr = NULL;
>    }
>
> Index: lx26-20-mm2b/kernel/nsproxy.c
> ===================================================================
> --- lx26-20-mm2b.orig/kernel/nsproxy.c 2007-03-09 19:00:14.000000000 -0800
> +++ lx26-20-mm2b/kernel/nsproxy.c 2007-03-09 19:01:09.000000000 -0800
> @@ -67,8 +67,6 @@ struct nsproxy *dup_nsproxy(struct nspro
>     get_uts_ns(ns->uts_ns);
>    if (ns->ipc_ns)
>     get_ipc_ns(ns->ipc_ns);
> -  if (ns->pid_ns)
> -   get_pid_ns(ns->pid_ns);
>   }
>
>   return ns;
> @@ -90,7 +88,7 @@ int copy_nsproxy(int flags, struct task_
>
>   get_nsproxy(old_ns);
>
> - ns_all = CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC | CLONE_NEWPID;
> + ns_all = CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC;
>
>   if (!(flags & ns_all))
>    return 0;
```

> @@ -115,17 +113,10 @@ int copy_nsproxy(int flags, struct task_
> if (err)
> goto out_ipc;
>
> - err = copy_pid_ns(flags, tsk);
> - if (err)
> - goto out_pid;
> -
> out:
> put_nsproxy(old_ns);
> return err;
>
> -out_pid:
> - if (new_ns->ipc_ns)
> - put_ipc_ns(new_ns->ipc_ns);
> out_ipc:
> if (new_ns->uts_ns)
> put_uts_ns(new_ns->uts_ns);
> Index: lx26-20-mm2b/kernel/pid.c
> ===
> --- lx26-20-mm2b.orig/kernel/pid.c 2007-03-09 19:00:42.000000000 -0800
> +++ lx26-20-mm2b/kernel/pid.c 2007-03-09 19:01:09.000000000 -0800
> @@ -221,8 +221,13 @@ fastcall void free_pid(struct pid *pid)
> hlist_del_rcu(&pid->pid_chain);
> spin_unlock_irqrestore(&pidmap_lock, flags);
>
> - hlist_for_each_entry(pid_nr, pos, &pid->pid_nrs, node)
> + hlist_for_each_entry(pid_nr, pos, &pid->pid_nrs, node) {
> free_pidmap(pid_nr->pid_ns, pid_nr->nr);
> +
> + /* put the reference we got in kref_init() in clone_pid_ns() */
> + if (pid_nr->nr == 1)
> + put_pid_ns(pid_nr->pid_ns);

Ok. This seems to make sense, but why restrict this to only pid 1?
I'm almost certain this will be the case, but... this seems a like
a unwarranted special case at the moment.

Basically why is it safe to restrict this to pid == 1. Is it possible
that we can race here?

> + }
> call_rcu(&pid->rcu, delayed_put_pid);
> }
>
> @@ -357,34 +362,48 @@ struct pid_namespace *pid_ns(struct pid
> return ns;
> }

>
> -struct pid *alloc_pid(void)
> +struct pid *alloc_pid(int flags)
> {
>   struct pid *pid;
>   enum pid_type type;
> - int nr = -1;
> - struct pid_nr *pid_nr;
> + struct pid_nr *pid_nr[2] = { NULL, NULL};
I would rather not see pid_nr special cased this way at all (a loop?)
but if we are going to I think two separate variables makes more
sense than this array.
> + struct pid_namespace *new_pid_ns = NULL;
>
>   pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
>   if (!pid)
>   return NULL;
>
> - nr = alloc_pidmap(task_pid_ns(current));
> - if (nr < 0)
> + pid_nr[0] = alloc_pidmap_pid_nr(task_pid_ns(current));
> + if (!pid_nr[0] < 0)
>   goto out_free_pid;
>
> - pid_nr = alloc_pid_nr(task_pid_ns(current));
> - if (!pid_nr)
> - goto out_free_pidmap;
> -
> + if (flags & CLONE_NEWPID) {
> + new_pid_ns = clone_pid_ns();
> + if (!new_pid_ns)
> +   goto out_free_pid_nr0;
> + /*
> +   * For now, allocate a pid_nr only for the new pid namespace.
> +   * Eventually we should allocate a pid_nr for each ancestor
> +   * namespace. While this could cost us additional memory in
> +   * deeply nested containers, it would allow us to see/signal
> +   * all processes from init-pid-ns.
> +   */
> + pid_nr[1] = alloc_pidmap_pid_nr(new_pid_ns);
> + if (!pid_nr[1])
> +   goto out_free_pid_ns;
> + }
>   atomic_set(&pid->count, 1);
> - pid->nr = pid_nr->nr = nr; /* pid->nr to be removed soon */
> + pid->nr = pid_nr[0]->nr; /* pid->nr to be removed soon */
>   for (type = 0; type < PIDTYPE_MAX; ++type)
>   INIT_HLIST_HEAD(&pid->tasks[type]);

>
> spin_lock_init(&pid->lock);
>
> INIT_HLIST_HEAD(&pid->pid_nrs);
> - hlist_add_head_rcu(&pid_nr->node, &pid->pid_nrs);
> + hlist_add_head_rcu(&pid_nr[0]->node, &pid->pid_nrs);
> +
> + if (pid_nr[1])
> + hlist_add_head_rcu(&pid_nr[1]->node, &pid->pid_nrs);
>
> spin_lock_irq(&pidmap_lock);
> hlist_add_head_rcu(&pid->pid_chain, &pid_hash[pid_hashfn(pid->nr)]);
> @@ -392,11 +411,15 @@ struct pid *alloc_pid(void)
>
> return pid;
>
> -out_free_pidmap:
> - free_pidmap(task_pid_ns(current), nr);
> +out_free_pid_ns:
> + put_pid_ns(new_pid_ns);
> +
> +out_free_pid_nr0:
> + free_pidmap_pid_nr(pid_nr[0]);
>
> out_free_pid:
> kmem_cache_free(pid_cachep, pid);
> +
> return NULL;
> }
>
> Index: lx26-20-mm2b/include/linux/pid_namespace.h
> ===
> --- lx26-20-mm2b.orig/include/linux/pid_namespace.h 2007-03-09
> 19:00:11.000000000 -0800
> +++ lx26-20-mm2b/include/linux/pid_namespace.h 2007-03-09 19:01:09.000000000
> -0800
> @@ -33,6 +33,7 @@ extern int unshare_pid_ns(unsigned long
> struct pid_nr **new_pid_nr);
> extern int copy_pid_ns(int flags, struct task_struct *tsk);
> extern void free_pid_ns(struct kref *kref);
> +extern struct pid_namespace *pid_ns(struct pid * pid);
>
> static inline void put_pid_ns(struct pid_namespace *ns)
> {
> @@ -41,18 +42,12 @@ static inline void put_pid_ns(struct pid
>
> static inline struct pid_namespace *task_pid_ns(struct task_struct *tsk)
> {

```
> -        return tsk->nsproxy->pid_ns;
> -}
> -
> -static inline void set_task_pid_ns(struct task_struct *tsk,
> -    struct pid_namespace * ns)
> -{
> -        tsk->nsproxy->pid_ns = ns
> +         return pid_ns(task_pid(tsk));
>  }
>
>  static inline struct task_struct *child_reaper(struct task_struct *tsk)
>  {
> - return init_pid_ns.child_reaper;
> + return task_pid_ns(tsk)->child_reaper;
>  }
>
>  #endif /* _LINUX_PID_NS_H */
```

Eric

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers