
Subject: [PATCH 13/16] Switch to operating with pid_numbers instead of pids
Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:10:47 GMT
[View Forum Message](#) <> [Reply to Message](#)

Make alloc_pid() initialize pid_numbers and hash them
into the hashtable, not the struct pid itself.

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

pid.c | 47 ++++++-----
1 files changed, 33 insertions(+), 14 deletions(-)

--- ./kernel/pid.c.ve12 2007-07-05 11:06:41.000000000 +0400

+++ ./kernel/pid.c 2007-07-05 11:08:23.000000000 +0400

@@ -28,8 +28,10 @@

#include <linux/hash.h>

#include <linux/pid_namespace.h>

#include <linux/init_task.h>

+#include <linux/proc_fs.h>

-#define pid_hashfn(nr) hash_long((unsigned long)nr, pidhash_shift)

+#define pid_hashfn(nr, ns) \

+ hash_long((unsigned long)nr + (unsigned long)ns, pidhash_shift)

static struct hlist_head *pid_hash;

static int pidhash_shift;

struct pid init_struct_pid = INIT_STRUCT_PID;

@@ -194,7 +198,7 @@ fastcall void put_pid(struct pid *pid)

if (!pid)

return;

- ns = pid->numbers[0].ns;

+ ns = pid->numbers[pid->level].ns;

if ((atomic_read(&pid->count) == 1) ||

atomic_dec_and_test(&pid->count))

kmem_cache_free(ns->pid_cachep, pid);

@@ -210,13 +214,17 @@ static void delayed_put_pid(struct rcu_h

fastcall void free_pid(struct pid *pid)

{

/* We can be called with write_lock_irq(&tasklist_lock) held */

+ int i;

unsigned long flags;

spin_lock_irqsave(&pidmap_lock, flags);

- hlist_del_rcu(&pid->pid_chain);

+ for (i = 0; i <= pid->level; i++)

+ hlist_del_rcu(&pid->numbers[i].pid_chain);

```

spin_unlock_irqrestore(&pidmap_lock, flags);

- free_pidmap(&init_pid_ns, pid->nr);
+ for (i = 0; i <= pid->level; i++)
+ free_pidmap(pid->numbers[i].ns, pid->numbers[i].nr);
+
+ call_rcu(&pid->rcu, delayed_put_pid);
}

@@ -224,30 +232,43 @@ struct pid *alloc_pid(struct pid_namespace
{
    struct pid *pid;
    enum pid_type type;
- int nr = -1;
+ struct pid_namespace *ns;
+ int i, nr;

- pid = kmem_cache_alloc(init_pid_ns.pid_cachep, GFP_KERNEL);
+ pid = kmem_cache_alloc(pid_ns->pid_cachep, GFP_KERNEL);
    if (!pid)
        goto out;

- nr = alloc_pidmap(current->nsproxy->pid_ns);
- if (nr < 0)
- goto out_free;
+ ns = pid_ns;
+ for (i = pid_ns->level; i >= 0; i--) {
+ nr = alloc_pidmap(ns);
+ if (nr < 0)
+ goto out_free;

+ pid->numbers[i].nr = nr;
+ pid->numbers[i].ns = ns;
+ ns = ns->parent;
+ }
+
+ pid->level = pid_ns->level;
+ atomic_set(&pid->count, 1);
- pid->nr = nr;
+ for (type = 0; type < PIDTYPE_MAX; ++type)
+ INIT_HLIST_HEAD(&pid->tasks[type]);

    spin_lock_irq(&pidmap_lock);
- hlist_add_head_rcu(&pid->pid_chain, &pid_hash[pid_hashfn(pid->nr)]);
+ for (i = pid->level; i >= 0; i--)
+ hlist_add_head_rcu(&pid->numbers[i].pid_chain,
+ &pid_hash[pid_hashfn(pid->numbers[i].nr,
+ pid->numbers[i].ns)]);

```

```

spin_unlock_irq(&pidmap_lock);
-
out:
return pid;

out_free:
- kmem_cache_free(init_pid_ns.pid_cachep, pid);
+ for (i++; i <= pid->level; i++)
+ free_pidmap(pid->numbers[i].ns, pid->numbers[i].nr);
+
+ kmem_cache_free(pid_ns->pid_cachep, pid);
pid = NULL;
goto out;
}
@@ -258,7 +279,7 @@ struct pid * fastcall find_pid_ns(int nr
struct pid_number *pnr;

hlist_for_each_entry_rcu(pnr, elem,
- &pid_hash[pid_hashfn(nr)], pid_chain)
+ &pid_hash[pid_hashfn(nr, ns)], pid_chain)
if (pnr->nr == nr && pnr->ns == ns)
return container_of(pnr, struct pid,
numbers[ns->level]);

```

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