

---

Subject: [PATCH 0/16] Pid namespaces

Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:01:59 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

This is "submission for inclusion" of hierarchical, not kconfig configurable, zero overheaded ;) pid namespaces.

The overall idea is the following:

The namespace are organized as a tree - once a task is cloned with CLONE\_NEWPIDS (yes, I've also switched to it :) the new namespace becomes the parent's child and tasks living in the parent namespace see the tasks from the new one. The numerical ids are used on the kernel-user boundary, i.e. when we export pid to user we show the id, that should be used to address the task in question from the namespace we're exporting this id to.

The main difference from Suka's patches are the following:

0. Suka's patches change the kernel/pid.c code too heavy. This set keeps the kernel code look like it was without the patches. However, this is a minor issue. The major is:

1. Suka's approach is to remove the notion of the task's numerical pid from the kernel at all. The numbers are used on the kernel-user boundary or within the kernel but with the namespace this nr belongs to. This results in massive changes of struct's members fro int pid to struct pid \*pid, task->pid becomes the virtual id and so on and so forth.

My approach is to keep the good old logic in the kernel. The task->pid is a global and unique pid, find\_pid() finds the pid by its global id and so on. The virtual ids appear on the user-kernel boundary only. Thus drivers and other kernel code may still be unaware of pids unless they do not communicate with the userspace and get/put numerical pids.

And some more minor differences:

2. Suka's patches have the limit of pid namespace nesting. My patches do not.

3. Suka assumes that pid namespace can live without proc mount and tries to make the code work with pid\_ns->proc\_mnt change from NULL to not-NULL from times to times.

My code calls the kern\_mount() at the namespace creation and thus the pid\_namespace always works with proc.

There are some small issues that I can describe if someone is interested.

The tests like nptl perf, unixbench spawn, getpid and others didn't reveal any performance degradation in init\_namespace with the RHEL5 kernel .config file. I admit, that different .config-s may show that patches hurt the performance, but the intention was \*not\* to make the kernel work worse with popular distributions.

This set has some ways to move forward, but this is some kind of a core, that do not change the init\_pid\_namespace behavior (checked with LTP tests) and may require some hacking to do with the namespaces only.

Patches apply to 2.6.22-rc6-mm1.

---

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

---

---

Subject: [PATCH 1/16] Round up the API  
Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:03:17 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

The set of functions process\_session, task\_session, process\_group and task\_pgrp is confusing, as the names can be mixed with each other when looking at the code for a long time.

The proposals are to

- \* equip the functions that return the integer with \_nr suffix to represent that fact,
- \* and to make all functions work with task (not process) by making the common prefix of the same name.

For monotony the routines signal\_session() and set\_signal\_session() are replaced with task\_session\_nr() and set\_task\_session(), especially since they are only used with the explicit task->signal dereference.

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

Acked-by: Serge E. Hallyn <serue@us.ibm.com>

---

```
arch/mips/kernel/irixelf.c | 4 +++-
arch/mips/kernel/irixsig.c | 2 +-
arch/mips/kernel/sysirix.c | 4 +++-
```

```

arch/sparc64/solaris/misc.c | 4 +++-
drivers/char/tty_io.c      | 4 +++-
fs/autofs/inode.c         | 2 +-
fs/autofs/root.c          | 4 +++-
fs/autofs4/autofs_i.h     | 2 +-
fs/autofs4/inode.c        | 4 +++-
fs/autofs4/root.c         | 4 +++-
fs/binfmt_elf.c           | 8 ++++-----
fs/binfmt_elf_fdpic.c     | 8 ++++-----
fs/coda/upcall.c          | 2 +-
fs/proc/array.c           | 4 +++-
include/linux/sched.h     | 15 ++++-----
kernel/exit.c             | 10 ++++-----
kernel/fork.c             | 4 +++-
kernel/signal.c           | 2 +-
kernel/sys.c              | 14 ++++-----
19 files changed, 48 insertions(+), 53 deletions(-)

```

```

--- ./arch/mips/kernel/irixelf.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./arch/mips/kernel/irixelf.c 2007-06-14 15:52:54.000000000 +0400
@@ -1170,8 +1170,8 @@ static int irix_core_dump(long signr, st
    prstatus.pr_sighold = current->blocked.sig[0];
    psinfo.pr_pid = prstatus.pr_pid = current->pid;
    psinfo.pr_ppid = prstatus.pr_ppid = current->parent->pid;
- psinfo.pr_pgrp = prstatus.pr_pgrp = process_group(current);
- psinfo.pr_sid = prstatus.pr_sid = process_session(current);
+ psinfo.pr_pgrp = prstatus.pr_pgrp = task_pgrp_nr(current);
+ psinfo.pr_sid = prstatus.pr_sid = task_session_nr(current);
    if (current->pid == current->tgid) {
/*
 * This is the record for the group leader. Add in the
--- ./arch/mips/kernel/irixsig.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./arch/mips/kernel/irixsig.c 2007-06-14 15:52:54.000000000 +0400
@@ -609,7 +609,7 @@ repeat:
    p = list_entry(_p,struct task_struct,sibling);
    if ((type == IRIX_P_PID) && p->pid != pid)
        continue;
- if ((type == IRIX_P_PGID) && process_group(p) != pid)
+ if ((type == IRIX_P_PGID) && task_pgrp_nr(p) != pid)
        continue;
    if ((p->exit_signal != SIGCHLD))
        continue;
--- ./arch/mips/kernel/sysirix.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./arch/mips/kernel/sysirix.c 2007-06-14 15:52:54.000000000 +0400
@@ -763,11 +763,11 @@ asmlinkage int irix_setpgrp(int flags)
    printk("[%s:%d] setpgrp(%d) ", current->comm, current->pid, flags);
#endif
    if(!flags)

```

```

- error = process_group(current);
+ error = task_pgrp_nr(current);
  else
    error = sys_setsid();
#ifdef DEBUG_PROCGRPS
- printk("returning %d\n", process_group(current));
+ printk("returning %d\n", task_pgrp_nr(current));
#endif

  return error;
--- ./arch/sparc64/solaris/misc.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./arch/sparc64/solaris/misc.c 2007-06-14 15:52:54.000000000 +0400
@@ -415,7 +415,7 @@ asmlinkage int solaris_procids(int cmd,

  switch (cmd) {
  case 0: /* getpgrp */
- return process_group(current);
+ return task_pgrp_nr(current);
  case 1: /* setpgrp */
  {
    int (*sys_setpgid)(pid_t,pid_t) =
@@ -426,7 +426,7 @@ asmlinkage int solaris_procids(int cmd,
    ret = sys_setpgid(0, 0);
    if (ret) return ret;
    proc_clear_tty(current);
- return process_group(current);
+ return task_pgrp_nr(current);
  }
  case 2: /* getsid */
  {
--- ./drivers/char/tty_io.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./drivers/char/tty_io.c 2007-06-14 15:52:55.000000000 +0400
@@ -3483,7 +3483,7 @@ void __do_SAK(struct tty_struct *tty)
/* Kill the entire session */
do_each_pid_task(session, PIDTYPE_SID, p) {
  printk(KERN_NOTICE "SAK: killed process %d"
- " (%s): process_session(p)==tty->session\n",
+ " (%s): task_session_nr(p)==tty->session\n",
    p->pid, p->comm);
  send_sig(SIGKILL, p, 1);
} while_each_pid_task(session, PIDTYPE_SID, p);
@@ -3493,7 +3493,7 @@ void __do_SAK(struct tty_struct *tty)
do_each_thread(g, p) {
  if (p->signal->tty == tty) {
    printk(KERN_NOTICE "SAK: killed process %d"
- " (%s): process_session(p)==tty->session\n",
+ " (%s): task_session_nr(p)==tty->session\n",
    p->pid, p->comm);

```

```

    send_sig(SIGKILL, p, 1);
    continue;
--- ./fs/autofs/inode.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./fs/autofs/inode.c 2007-06-14 15:52:55.000000000 +0400
@@ -80,7 +80,7 @@ static int parse_options(char *options,

    *uid = current->uid;
    *gid = current->gid;
- *pgrp = process_group(current);
+ *pgrp = task_pgrp_nr(current);

    *minproto = *maxproto = AUTOFS_PROTO_VERSION;

--- ./fs/autofs/root.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./fs/autofs/root.c 2007-06-14 15:52:55.000000000 +0400
@@ -215,7 +215,7 @@ static struct dentry *autofs_root_lookup
    oz_mode = autofs_oz_mode(sbi);
    DPRINTK(("autofs_lookup: pid = %u, pgrp = %u, catatonic = %d, "
        "oz_mode = %d\n", pid_nr(task_pid(current)),
-    process_group(current), sbi->catatonic,
+    task_pgrp_nr(current), sbi->catatonic,
        oz_mode));

/*
@@ -536,7 +536,7 @@ static int autofs_root_ioctl(struct inode
    struct autofs_sb_info *sbi = autofs_sb_info(inode->i_sb);
    void __user *argp = (void __user *)arg;

- DPRINTK(("autofs_ioctl: cmd = 0x%08x, arg = 0x%08lx, sbi = %p, pgrp =
%u\n",cmd,arg,sbi,process_group(current)));
+ DPRINTK(("autofs_ioctl: cmd = 0x%08x, arg = 0x%08lx, sbi = %p, pgrp =
%u\n",cmd,arg,sbi,task_pgrp_nr(current)));

    if (_IOC_TYPE(cmd) != _IOC_TYPE(AUTOFS_IOC_FIRST) ||
        _IOC_NR(cmd) - _IOC_NR(AUTOFS_IOC_FIRST) >= AUTOFS_IOC_COUNT)
--- ./fs/autofs4/autofs_i.h.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./fs/autofs4/autofs_i.h 2007-06-14 15:52:55.000000000 +0400
@@ -131,7 +131,7 @@ static inline struct autofs_info *autofs
    filesystem without "magic.") */

static inline int autofs4_oz_mode(struct autofs_sb_info *sbi) {
- return sbi->catatonic || process_group(current) == sbi->oz_pgrp;
+ return sbi->catatonic || task_pgrp_nr(current) == sbi->oz_pgrp;
}

/* Does a dentry have some pending activity? */
--- ./fs/autofs4/inode.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./fs/autofs4/inode.c 2007-06-14 15:52:55.000000000 +0400

```

```

@@ -226,7 +226,7 @@ static int parse_options(char *options,

*uid = current->uid;
*gid = current->gid;
- *pgrp = process_group(current);
+ *pgrp = task_pgrp_nr(current);

*minproto = AUTOFS_MIN_PROTO_VERSION;
*maxproto = AUTOFS_MAX_PROTO_VERSION;
@@ -325,7 +325,7 @@ int autofs4_fill_super(struct super_bloc
sbi->pipe = NULL;
sbi->catatonic = 1;
sbi->exp_timeout = 0;
- sbi->oz_pgrp = process_group(current);
+ sbi->oz_pgrp = task_pgrp_nr(current);
sbi->sb = s;
sbi->version = 0;
sbi->sub_version = 0;
--- ./fs/autofs4/root.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./fs/autofs4/root.c 2007-06-14 15:52:55.000000000 +0400
@@ -582,7 +582,7 @@ static struct dentry *autofs4_lookup(str
oz_mode = autofs4_oz_mode(sbi);

DPRINTK("pid = %u, pgrp = %u, catatonic = %d, oz_mode = %d",
- current->pid, process_group(current), sbi->catatonic, oz_mode);
+ current->pid, task_pgrp_nr(current), sbi->catatonic, oz_mode);

unhashed = autofs4_lookup_unhashed(sbi, dentry->d_parent, &dentry->d_name);
if (!unhashed) {
@@ -973,7 +973,7 @@ static int autofs4_root_ioctl(struct ino
void __user *p = (void __user *)arg;

DPRINTK("cmd = 0x%08x, arg = 0x%08lx, sbi = %p, pgrp = %u",
- cmd,arg,sbi,process_group(current));
+ cmd,arg,sbi,task_pgrp_nr(current));

if (_IOC_TYPE(cmd) != _IOC_TYPE(AUTOFS_IOC_FIRST) ||
    _IOC_NR(cmd) - _IOC_NR(AUTOFS_IOC_FIRST) >= AUTOFS_IOC_COUNT)
--- ./fs/binfmt_elf.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./fs/binfmt_elf.c 2007-06-14 15:52:55.000000000 +0400
@@ -1394,8 +1394,8 @@ static void fill_prstatus(struct elf_prs
prstatus->pr_sighold = p->blocked.sig[0];
prstatus->pr_pid = p->pid;
prstatus->pr_ppid = p->parent->pid;
- prstatus->pr_pgrp = process_group(p);
- prstatus->pr_sid = process_session(p);
+ prstatus->pr_pgrp = task_pgrp_nr(p);
+ prstatus->pr_sid = task_session_nr(p);

```

```

if (thread_group_leader(p)) {
/*
 * This is the record for the group leader. Add in the
@@ -1440,8 +1440,8 @@ static int fill_psinfo(struct elf_prpsin

psinfo->pr_pid = p->pid;
psinfo->pr_ppid = p->parent->pid;
- psinfo->pr_pgrp = process_group(p);
- psinfo->pr_sid = process_session(p);
+ psinfo->pr_pgrp = task_pgrp_nr(p);
+ psinfo->pr_sid = task_session_nr(p);

i = p->state ? ffz(~p->state) + 1 : 0;
psinfo->pr_state = i;
--- ./fs/binfmt_elf_fdpic.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./fs/binfmt_elf_fdpic.c 2007-06-14 15:52:55.000000000 +0400
@@ -1344,8 +1344,8 @@ static void fill_prstatus(struct elf_prs
prstatus->pr_sighold = p->blocked.sig[0];
prstatus->pr_pid = p->pid;
prstatus->pr_ppid = p->parent->pid;
- prstatus->pr_pgrp = process_group(p);
- prstatus->pr_sid = process_session(p);
+ prstatus->pr_pgrp = task_pgrp_nr(p);
+ prstatus->pr_sid = task_session_nr(p);
if (thread_group_leader(p)) {
/*
 * This is the record for the group leader. Add in the
@@ -1393,8 +1393,8 @@ static int fill_psinfo(struct elf_prpsin

psinfo->pr_pid = p->pid;
psinfo->pr_ppid = p->parent->pid;
- psinfo->pr_pgrp = process_group(p);
- psinfo->pr_sid = process_session(p);
+ psinfo->pr_pgrp = task_pgrp_nr(p);
+ psinfo->pr_sid = task_session_nr(p);

i = p->state ? ffz(~p->state) + 1 : 0;
psinfo->pr_state = i;
--- ./fs/coda/upcall.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./fs/coda/upcall.c 2007-06-14 15:52:55.000000000 +0400
@@ -53,7 +53,7 @@ static void *alloc_upcall(int opcode, in

inp->ih.opcode = opcode;
inp->ih.pid = current->pid;
- inp->ih.pgid = process_group(current);
+ inp->ih.pgid = task_pgrp_nr(current);
#ifdef CONFIG_CODA_FS_OLD_API
memset(&inp->ih.cred, 0, sizeof(struct coda_cred));

```

```

inp->ih.cred.cr_fsuid = current->fsuid;
--- ./fs/proc/array.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./fs/proc/array.c 2007-06-14 15:52:55.000000000 +0400
@@ -414,8 +414,8 @@ static int do_task_stat(struct task_stru
    stime += cputime_to_clock_t(sig->stime);
}

- sid = signal_session(sig);
- pgid = process_group(task);
+ sid = task_session_nr(task);
+ pgid = task_pgrp_nr(task);
  ppid = rcu_dereference(task->real_parent)->tgid;

  unlock_task_sighand(task, &flags);
--- ./include/linux/sched.h.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./include/linux/sched.h 2007-06-14 15:52:55.000000000 +0400
@@ -1153,24 +1153,19 @@ struct task_struct {
 #endif
};

-static inline pid_t process_group(struct task_struct *tsk)
+static inline pid_t task_pgrp_nr(struct task_struct *tsk)
{
  return tsk->signal->pgrp;
}

-static inline pid_t signal_session(struct signal_struct *sig)
-{-
- return sig->__session;
-}
-
-static inline pid_t process_session(struct task_struct *tsk)
+static inline pid_t task_session_nr(struct task_struct *tsk)
{
- return signal_session(tsk->signal);
+ return tsk->signal->__session;
}

-static inline void set_signal_session(struct signal_struct *sig, pid_t session)
+static inline void set_task_session(struct task_struct *tsk, pid_t session)
{
- sig->__session = session;
+ tsk->signal->__session = session;
}

static inline struct pid *task_pid(struct task_struct *task)
--- ./kernel/exit.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./kernel/exit.c 2007-06-14 15:52:55.000000000 +0400

```



```

@@ -309,12 +309,12 @@ void __set_special_pids(pid_t session, p
{
    struct task_struct *curr = current->group_leader;

- if (process_session(curr) != session) {
+ if (task_session_nr(curr) != session) {
    detach_pid(curr, PIDTYPE_SID);
- set_signal_session(curr->signal, session);
+ set_task_session(curr, session);
    attach_pid(curr, PIDTYPE_SID, find_pid(session));
}
- if (process_group(curr) != pgrp) {
+ if (task_pgrp_nr(curr) != pgrp) {
    detach_pid(curr, PIDTYPE_PGID);
    curr->signal->pgrp = pgrp;
    attach_pid(curr, PIDTYPE_PGID, find_pid(pgrp));
@@ -1055,10 +1055,10 @@ static int eligible_child(pid_t pid, int
    if (p->pid != pid)
        return 0;
    } else if (!pid) {
- if (process_group(p) != process_group(current))
+ if (task_pgrp_nr(p) != task_pgrp_nr(current))
        return 0;
    } else if (pid != -1) {
- if (process_group(p) != -pid)
+ if (task_pgrp_nr(p) != -pid)
        return 0;
    }
}

```

```

--- ./kernel/fork.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./kernel/fork.c 2007-06-14 15:52:55.000000000 +0400
@@ -1259,8 +1259,8 @@ static struct task_struct *copy_process(

```

```

    if (thread_group_leader(p)) {
        p->signal->tty = current->signal->tty;
- p->signal->pgrp = process_group(current);
- set_signal_session(p->signal, process_session(current));
+ 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));
    }

```

```

--- ./kernel/signal.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./kernel/signal.c 2007-06-14 15:52:55.000000000 +0400
@@ -517,7 +517,7 @@ static int check_kill_permission(int sig
    error = -EPERM;
    if ((info == SEND_SIG_NOINFO || (!is_si_special(info) && SI_FROMUSER(info)))
        && ((sig != SIGCONT) ||

```

```

- (process_session(current) != process_session(t))
+ (task_session_nr(current) != task_session_nr(t))
  && (current->euid ^ t->suid) && (current->euid ^ t->uid)
  && (current->uid ^ t->suid) && (current->uid ^ t->uid)
  && !capable(CAP_KILL))
--- ./kernel/sys.c.apiren 2007-06-14 12:14:29.000000000 +0400
+++ ./kernel/sys.c 2007-06-14 15:52:55.000000000 +0400
@@ -1485,7 +1485,7 @@ asmlinkage long sys_setpgid(pid_t pid, p
 if (err)
 goto out;

- if (process_group(p) != pgid) {
+ if (task_pgrp_nr(p) != pgid) {
 detach_pid(p, PIDTYPE_PGID);
 p->signal->pgrp = pgid;
 attach_pid(p, PIDTYPE_PGID, find_pid(pgid));
@@ -1501,7 +1501,7 @@ out:
asmlinkage long sys_getpgid(pid_t pid)
{
 if (!pid)
- return process_group(current);
+ return task_pgrp_nr(current);
 else {
 int retval;
 struct task_struct *p;
@@ -1513,7 +1513,7 @@ asmlinkage long sys_getpgid(pid_t pid)
 if (p) {
 retval = security_task_getpgid(p);
 if (!retval)
- retval = process_group(p);
+ retval = task_pgrp_nr(p);
 }
 read_unlock(&tasklist_lock);
 return retval;
@@ -1525,7 +1525,7 @@ asmlinkage long sys_getpgid(pid_t pid)
asmlinkage long sys_getpgrp(void)
{
 /* SMP - assuming writes are word atomic this is fine */
- return process_group(current);
+ return task_pgrp_nr(current);
}

#endif
@@ -1533,7 +1533,7 @@ asmlinkage long sys_getpgrp(void)
asmlinkage long sys_getsid(pid_t pid)
{
 if (!pid)
- return process_session(current);

```

```

+ return task_session_nr(current);
  else {
    int retval;
    struct task_struct *p;
@@ -1545,7 +1545,7 @@ asmlinkage long sys_getsid(pid_t pid)
    if (p) {
      retval = security_task_getsid(p);
      if (!retval)
-     retval = process_session(p);
+     retval = task_session_nr(p);
    }
    read_unlock(&tasklist_lock);
    return retval;
@@ -1582,7 +1582,7 @@ asmlinkage long sys_setsid(void)
    group_leader->signal->tty = NULL;
    spin_unlock(&group_leader->sigband->siglock);

- err = process_group(group_leader);
+ err = task_pgrp_nr(group_leader);
out:
  write_unlock_irq(&tasklist_lock);
  return err;

```

---

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

---



---

Subject: [PATCH 2/16] Miscellaneous preparations for namespaces  
 Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:03:54 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

The most important change is moving `exit_task_namespaces()` inside `exit_notify()` to make it possible to notify the exiting task's parent. However this should be done before `release_task()` to address the issue pointed by Sukadev with NFS kernel thread.

Other changes are small and do not deserve separate description.

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

---

```

include/linux/pid_namespace.h | 7 +++++-
kernel/exit.c                 | 3 +-
kernel/pid.c                   | 2 ++
3 files changed, 8 insertions(+), 4 deletions(-)

```

```

--- ./include/linux/pid_namespace.h.ve1 2007-07-06 10:58:57.000000000 +0400
+++ ./include/linux/pid_namespace.h 2007-07-06 11:03:18.000000000 +0400
@@ -4,7 +4,6 @@
#include <linux/sched.h>
#include <linux/mm.h>
#include <linux/threads.h>
-#include <linux/pid.h>
#include <linux/nsproxy.h>
#include <linux/kref.h>

@@ -24,9 +23,10 @@ struct pid_namespace {

extern struct pid_namespace init_pid_ns;

-static inline void get_pid_ns(struct pid_namespace *ns)
+static inline struct pid_namespace *get_pid_ns(struct pid_namespace *ns)
{
    kref_get(&ns->kref);
+ return ns;
}

extern struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *ns);
@@ -39,7 +39,8 @@ static inline void put_pid_ns(struct pid

static inline struct task_struct *child_reaper(struct task_struct *tsk)
{
- return init_pid_ns.child_reaper;
+ BUG_ON(tsk != current);
+ return tsk->nsproxy->pid_ns->child_reaper;
}

#endif /* _LINUX_PID_NS_H */
--- ./kernel/exit.c.ve1 2007-07-06 11:02:55.000000000 +0400
+++ ./kernel/exit.c 2007-07-06 11:02:55.000000000 +0400
@@ -862,6 +862,8 @@ static void exit_notify(struct task_stru
    release_task(t);
}

+ exit_task_namespaces(tsk);
+
/* If the process is dead, release it - nobody will wait for it */
if (state == EXIT_DEAD)
    release_task(tsk);
@@ -1002,7 +1004,6 @@ fastcall NORET_TYPE void do_exit(long co

    tsk->exit_code = code;
    proc_exit_connector(tsk);

```

```

- exit_task_namespaces(tsk);
  exit_notify(tsk);
  #ifdef CONFIG_NUMA
    mpol_free(tsk->mempolicy);
--- ./kernel/pid.c.ve1 2007-07-06 10:58:57.000000000 +0400
+++ ./kernel/pid.c 2007-07-06 11:02:55.000000000 +0400
@@ -71,6 +71,8 @@ struct pid_namespace init_pid_ns = {
    .child_reaper = &init_task
};

+EXPORT_SYMBOL_GPL(init_pid_ns);
+
+/*
+ * Note: disable interrupts while the pidmap_lock is held as an
+ * interrupt might come in and do read_lock(&tasklist_lock).

```

---

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

---



---

Subject: [PATCH 3/16] Introduce MS\_KERNMOUNT flag  
Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:04:28 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

This flag tells the .get\_sb callback that this is a kern\_mount() call so that it can trust \*data pointer to be valid in-kernel one.

Running a few steps forward - this will be needed for proc to create the superblock and store a valid pid namespace on it during the namespace creation. The reason, why the namespace cannot live without proc mount is described in the appropriate patch.

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

---

```

fs/namespace.c | 3 +-
fs/super.c     | 6 +++---
include/linux/fs.h | 4 +++-
3 files changed, 8 insertions(+), 5 deletions(-)

```

```

diff -upr linux-2.6.22-rc4-mm2.orig/fs/namespace.c linux-2.6.22-rc4-mm2-2/fs/namespace.c
--- linux-2.6.22-rc4-mm2.orig/fs/namespace.c 2007-06-14 12:00:06.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/fs/namespace.c 2007-07-04 19:00:39.000000000 +0400
@@ -1558,7 +1558,8 @@ long do_mount(char *dev_name, char *dir_
    mnt_flags |= MNT_NOMNT;

```

```
flags &= ~(MS_NOSUID | MS_NOEXEC | MS_NODEV | MS_ACTIVE |
- MS_NOATIME | MS_NODIRATIME | MS_RELATIME | MS_NOMNT);
+ MS_NOATIME | MS_NODIRATIME | MS_RELATIME |
+ MS_NOMNT | MS_KERNMOUNT);
```

```
/* ... and get the mountpoint */
retval = path_lookup(dir_name, LOOKUP_FOLLOW, &nd);
diff -upr linux-2.6.22-rc4-mm2.orig/fs/super.c linux-2.6.22-rc4-mm2-2/fs/super.c
--- linux-2.6.22-rc4-mm2.orig/fs/super.c 2007-06-07 15:37:30.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/fs/super.c 2007-07-04 19:00:39.000000000 +0400
@@ -942,9 +942,9 @@ do_kern_mount(const char *fstype, int fl
return mnt;
}
```

```
-struct vfsmount *kern_mount(struct file_system_type *type)
+struct vfsmount *kern_mount_data(struct file_system_type *type, void *data)
{
- return vfs_kern_mount(type, 0, type->name, NULL);
+ return vfs_kern_mount(type, MS_KERNMOUNT, type->name, data);
}
```

```
-EXPORT_SYMBOL(kern_mount);
+EXPORT_SYMBOL_GPL(kern_mount_data);
diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/fs.h linux-2.6.22-rc4-mm2-2/include/linux/fs.h
--- linux-2.6.22-rc4-mm2.orig/include/linux/fs.h 2007-06-14 12:00:06.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/include/linux/fs.h 2007-07-04 19:00:39.000000000 +0400
@@ -130,6 +130,7 @@ extern int dir_notify_enable;
#define MS_NO_LEASES (1<<22) /* fs does not support leases */
#define MS_SETUSER (1<<23) /* set mnt_uid to current user */
#define MS_NOMNT (1<<24) /* don't allow unprivileged submounts */
+#define MS_KERNMOUNT (1<<25) /* this is a kern_mount call */
#define MS_ACTIVE (1<<30)
#define MS_NOUSER (1<<31)
```

```
@@ -1490,7 +1491,8 @@ void unnamed_dev_init(void);
```

```
extern int register_filesystem(struct file_system_type *);
extern int unregister_filesystem(struct file_system_type *);
-extern struct vfsmount *kern_mount(struct file_system_type *);
+extern struct vfsmount *kern_mount_data(struct file_system_type *, void *data);
+#define kern_mount(type) kern_mount_data(type, NULL)
extern int may_umount_tree(struct vfsmount *);
extern int may_umount(struct vfsmount *);
extern void umount_tree(struct vfsmount *, int, struct list_head *);
```

---

Containers mailing list

---

Subject: [PATCH 4/16] Change data structures for pid namespaces  
Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:05:00 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

struct pid\_namespace will have the kmem\_cache to allocate the pids from, the parent, as they are hierarchical, and the level of nesting value.

struct pid will have a variable length array of pid\_number-s one for each namespace this pid lives in. The level value shows the level of the namespace this pid lives in and thus - the number of elements in the numbers array.

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

---

```
include/linux/init_task.h | 6 ++++++
include/linux/pid.h       | 9 ++++++++
include/linux/pid_namespace.h | 3 +++
kernel/pid.c             | 3 +-
4 files changed, 20 insertions(+), 1 deletion(-)
```

```
diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid.h linux-2.6.22-rc4-mm2-2/include/linux/pid.h
--- linux-2.6.22-rc4-mm2.orig/include/linux/pid.h 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/include/linux/pid.h 2007-07-04 19:00:38.000000000 +0400
@@ -40,6 +40,13 @@ enum pid_type
 * processes.
 */
```

```
+struct pid_number {
+ /* Try to keep pid_chain in the same cacheline as nr for find_pid */
+ int nr;
+ struct pid_namespace *ns;
+ struct hlist_node pid_chain;
+};
+
+struct pid
+{
+ atomic_t count;
@@ -40,6 +40,8 @@ enum pid_type
 /* lists of tasks that use this pid */
 struct hlist_head tasks[PIDTYPE_MAX];
 struct rcu_head rcu;
```

```

+ int level;
+ struct pid_number numbers[1];
};

extern struct pid init_struct_pid;
diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid_namespace.h
linux-2.6.22-rc4-mm2-2/include/linux/pid_namespace.h
--- linux-2.6.22-rc4-mm2.orig/include/linux/pid_namespace.h 2007-06-14 12:14:29.000000000
+0400
+++ linux-2.6.22-rc4-mm2-2/include/linux/pid_namespace.h 2007-07-04 19:00:39.000000000
+0400
@@ -16,7 +15,10 @@ struct pidmap {
    struct kref kref;
    struct pidmap pidmap[PIDMAP_ENTRIES];
    int last_pid;
+ int level;
    struct task_struct *child_reaper;
+ struct kmem_cache *pid_cache;
+ struct pid_namespace *parent;
};

extern struct pid_namespace init_pid_ns;
diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/init_task.h
linux-2.6.22-rc4-mm2-2/include/linux/init_task.h
--- linux-2.6.22-rc4-mm2.orig/include/linux/init_task.h 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/include/linux/init_task.h 2007-07-04 19:00:38.000000000 +0400
@@ -91,6 +91,12 @@ extern struct group_info init_groups;
    { .first = &init_task.pids[PIDTYPE_SID].node }, \
    }, \
    .rcu = RCU_HEAD_INIT, \
+ .level = 0, \
+ .numbers = { { \
+ .nr = 0, \
+ .ns = &init_pid_ns, \
+ .pid_chain = { .next = NULL, .pprev = NULL }, \
+ }, } \
}

#define INIT_PID_LINK(type) \
diff -upr linux-2.6.22-rc4-mm2.orig/kernel/pid.c linux-2.6.22-rc4-mm2-2/kernel/pid.c
--- linux-2.6.22-rc4-mm2.orig/kernel/pid.c 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/kernel/pid.c 2007-07-04 19:00:38.000000000 +0400
@@ -61,7 +62,8 @@ static inline int mk_pid(struct pid_name
    [ 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_GPL(init_pid_ns);
```

---

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

---

---

Subject: [PATCH 5/16] Make proc be mountable from different pid namespaces  
Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:05:42 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Each pid namespace should have the proc\_mnt pointer even when there's no user mounts to make proc\_flush\_task() work. To do this we call the kern\_mount() to obtain the proc mount point.

Since the current pid\_namespace during this call is not the newly created one we use the introduced MS\_KERNMOUNT flag to pass the namespace pointer to the proc\_get\_sb() call.

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

---

```
fs/proc/inode.c      | 20 ++++++
fs/proc/internal.h  | 2
fs/proc/root.c      | 116 ++++++
include/linux/pid_namespace.h | 3 +
include/linux/proc_fs.h | 15 +++++
5 files changed, 147 insertions(+), 9 deletions(-)
```

```
diff -upr linux-2.6.22-rc4-mm2.orig/fs/proc/inode.c linux-2.6.22-rc4-mm2-2/fs/proc/inode.c
--- linux-2.6.22-rc4-mm2.orig/fs/proc/inode.c 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/fs/proc/inode.c 2007-07-04 19:00:38.000000000 +0400
@@ -16,6 +16,7 @@
#include <linux/init.h>
#include <linux/module.h>
#include <linux/smp_lock.h>
+#include <linux/pid_namespace.h>

#include <asm/system.h>
#include <asm/uaccess.h>
@@ -429,9 +430,17 @@ out_mod:
return NULL;
}
```

```

-int proc_fill_super(struct super_block *s, void *data, int silent)
+int proc_fill_super(struct super_block *s, struct pid_namespace *ns)
{
    struct inode * root_inode;
+ struct proc_dir_entry * root_dentry;
+
+ root_dentry = &proc_root;
+ if (ns != &init_pid_ns) {
+ root_dentry = create_proc_root();
+ if (root_dentry == NULL)
+ goto out_no_de;
+ }

    s->s_flags |= MS_NODIRATIME | MS_NOSUID | MS_NOEXEC;
    s->s_blocksize = 1024;
@@ -440,8 +449,8 @@ int proc_fill_super(struct super_block *
    s->s_op = &proc_sops;
    s->s_time_gran = 1;

- de_get(&proc_root);
- root_inode = proc_get_inode(s, PROC_ROOT_INO, &proc_root);
+ de_get(root_dentry);
+ root_inode = proc_get_inode(s, PROC_ROOT_INO, root_dentry);
    if (!root_inode)
        goto out_no_root;
    root_inode->i_uid = 0;
@@ -452,9 +461,10 @@ int proc_fill_super(struct super_block *
    return 0;

out_no_root:
- printk("proc_read_super: get root inode failed\n");
    iput(root_inode);
- de_put(&proc_root);
+ de_put(root_dentry);
+out_no_de:
+ printk("proc_read_super: get root inode failed\n");
    return -ENOMEM;
}
MODULE_LICENSE("GPL");
diff -upr linux-2.6.22-rc4-mm2.orig/fs/proc/internal.h linux-2.6.22-rc4-mm2-2/fs/proc/internal.h
--- linux-2.6.22-rc4-mm2.orig/fs/proc/internal.h 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/fs/proc/internal.h 2007-07-04 19:00:38.000000000 +0400
@@ -71,3 +71,5 @@ static inline int proc_fd(struct inode *
{
    return PROC_I(inode)->fd;
}
+

```

```

+struct proc_dir_entry * create_proc_root(void);
diff -upr linux-2.6.22-rc4-mm2.orig/fs/proc/root.c linux-2.6.22-rc4-mm2-2/fs/proc/root.c
--- linux-2.6.22-rc4-mm2.orig/fs/proc/root.c 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/fs/proc/root.c 2007-07-04 19:00:39.000000000 +0400
@@ -18,32 +18,89 @@
#include <linux/bitops.h>
#include <linux/smp_lock.h>
#include <linux/mount.h>
+#include <linux/pid_namespace.h>

#include "internal.h"

struct proc_dir_entry *proc_net, *proc_net_stat, *proc_bus, *proc_root_fs, *proc_root_driver;

+static int proc_test_super(struct super_block *sb, void *data)
+{
+ return sb->s_fs_info == data;
+}
+
+static int proc_set_super(struct super_block *sb, void *data)
+{
+ struct pid_namespace *ns;
+
+ ns = (struct pid_namespace *)data;
+ sb->s_fs_info = get_pid_ns(ns);
+ return set_anon_super(sb, NULL);
+}
+
static int proc_get_sb(struct file_system_type *fs_type,
int flags, const char *dev_name, void *data, struct vfsmount *mnt)
{
+ int err;
+ struct super_block *sb;
+ struct pid_namespace *ns;
+ struct proc_inode *ei;
+
if (proc_mnt) {
/* Seed the root directory with a pid so it doesn't need
* to be special in base.c. I would do this earlier but
* the only task alive when /proc is mounted the first time
* is the init_task and it doesn't have any pids.
*/
- struct proc_inode *ei;
ei = PROC_I(proc_mnt->mnt_sb->s_root->d_inode);
if (!ei->pid)
ei->pid = find_get_pid(1);
}
- return get_sb_single(fs_type, flags, data, proc_fill_super, mnt);

```

```

+
+ if (flags & MS_KERNMOUNT)
+ ns = (struct pid_namespace *)data;
+ else
+ ns = current->nsproxy->pid_ns;
+
+ sb = sget(fs_type, proc_test_super, proc_set_super, ns);
+ if (IS_ERR(sb))
+ return PTR_ERR(sb);
+
+ if (!sb->s_root) {
+ sb->s_flags = flags;
+ err = proc_fill_super(sb, ns);
+ if (err) {
+ up_write(&sb->s_umount);
+ deactivate_super(sb);
+ return err;
+ }
+
+ ei = PROC_I(sb->s_root->d_inode);
+ if (!ei->pid)
+ ei->pid = find_get_pid(1);
+ sb->s_flags |= MS_ACTIVE;
+
+ mntput(ns->proc_mnt);
+ ns->proc_mnt = mnt;
+ }
+
+ return simple_set_mnt(mnt, sb);
+}
+
+static void proc_kill_sb(struct super_block *sb)
+{
+ struct pid_namespace *ns;
+
+ ns = (struct pid_namespace *)sb->s_fs_info;
+ kill_anon_super(sb);
+ if (ns != NULL)
+ put_pid_ns(ns);
+ }

static struct file_system_type proc_fs_type = {
    .name = "proc",
    .get_sb = proc_get_sb,
    .kill_sb = kill_anon_super,
    .kill_sb = proc_kill_sb,
};

```

```

void __init proc_root_init(void)
@@ -60,6 +117,7 @@ void __init proc_root_init(void)
    unregister_filesystem(&proc_fs_type);
    return;
}
+
proc_misc_init();
proc_net = proc_mkdir("net", NULL);
proc_net_stat = proc_mkdir("net/stat", NULL);
@@ -153,6 +211,58 @@ struct proc_dir_entry proc_root = {
    .parent = &proc_root,
};

+/*
+ * creates the proc root entry for different proc trees
+ */
+
+struct proc_dir_entry * create_proc_root(void)
+{
+ struct proc_dir_entry *de;
+
+ de = kzalloc(sizeof(struct proc_dir_entry), GFP_KERNEL);
+ if (de != NULL) {
+ de->low_ino = PROC_ROOT_INO;
+ de->namelen = 5;
+ de->name = "/proc";
+ de->mode = S_IFDIR | S_IRUGO | S_IXUGO;
+ de->nlink = 2;
+ de->proc_iops = &proc_root_inode_operations;
+ de->proc_fops = &proc_root_operations;
+ de->parent = de;
+ }
+ return de;
+}
+
+int pid_ns_prepare_proc(struct pid_namespace *ns)
+{
+ struct vfsmount *mnt;
+
+ mnt = kern_mount_data(&proc_fs_type, ns);
+ if (!IS_ERR(mnt))
+ /*
+ * do not save the reference from the proc super
+ * block to the namespace. otherwise we will get
+ * a circular reference ns->proc_mnt->mnt_sb->ns
+ */
+ put_pid_ns(ns);
+ return 0;

```

```

+}
+
+void pid_ns_release_proc(struct pid_namespace *ns)
+{
+ struct vfsmount *mnt;
+
+ mnt = ns->proc_mnt;
+ /*
+ * do not put the namespace reference as it wa not get in
+ * pid_ns_prepare_proc(). safe to set NULL here as this
+ * namespace is already dead and all the proc mounts are
+ * released so nobudo will se this super block
+ */
+ mnt->mnt_sb->s_fs_info = NULL;
+ mntput(mnt);
+}
+
+ EXPORT_SYMBOL(proc_symlink);
+ EXPORT_SYMBOL(proc_mkdir);
+ EXPORT_SYMBOL(create_proc_entry);
diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid_namespace.h
linux-2.6.22-rc4-mm2-2/include/linux/pid_namespace.h
--- linux-2.6.22-rc4-mm2.orig/include/linux/pid_namespace.h 2007-06-14 12:14:29.000000000
+0400
+++ linux-2.6.22-rc4-mm2-2/include/linux/pid_namespace.h 2007-07-04 19:00:39.000000000
+0400
@@ -16,6 +15,9 @@ struct pidmap {
    struct task_struct *child_reaper;
    struct kmem_cache *pid_cache;
    struct pid_namespace *parent;
+#ifdef CONFIG_PROC_FS
+ struct vfsmount *proc_mnt;
+#endif
};

extern struct pid_namespace init_pid_ns;
diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/proc_fs.h
linux-2.6.22-rc4-mm2-2/include/linux/proc_fs.h
--- linux-2.6.22-rc4-mm2.orig/include/linux/proc_fs.h 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/include/linux/proc_fs.h 2007-07-04 19:00:38.000000000 +0400
@@ -126,7 +126,8 @@ extern struct proc_dir_entry *create_pro
extern void remove_proc_entry(const char *name, struct proc_dir_entry *parent);

extern struct vfsmount *proc_mnt;
-extern int proc_fill_super(struct super_block *,void *,int);
+struct pid_namespace;
+extern int proc_fill_super(struct super_block *, struct pid_namespace *);
extern struct inode *proc_get_inode(struct super_block *, unsigned int, struct proc_dir_entry *);

```

```

/*
@@ -143,6 +144,9 @@ extern const struct file_operations proc
extern const struct file_operations proc_kmsg_operations;
extern const struct file_operations ppc_htab_operations;

+extern int pid_ns_prepare_proc(struct pid_namespace *ns);
+extern void pid_ns_release_proc(struct pid_namespace *ns);
+
/*
 * proc_tty.c
 */
@@ -248,6 +254,15 @@ static inline void proc_tty_unregister_d

extern struct proc_dir_entry proc_root;

+static inline int pid_ns_prepare_proc(struct pid_namespace *ns)
+{
+ return 0;
+}
+
+static inline void pid_ns_release_proc(struct pid_namespace *ns)
+{
+}
+
#endif /* CONFIG_PROC_FS */

#if !defined(CONFIG_PROC_KCORE)

```

---

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

---



---

Subject: [PATCH 6/16] Helpers to obtain pid numbers  
Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:06:22 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

When showing pid to user or getting the pid numerical id for in-kernel use the value of this id may differ depending on the namespace.

This set of helpers is used to get the global pid nr, the virtual (i.e. seen by task in its namespace) nr and the nr as it is seen from the specified namespace.

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

---

```
include/linux/pid.h | 27 ++++++++
include/linux/sched.h | 108 ++++++++
kernel/pid.c | 8 +++
3 files changed, 132 insertions(+), 11 deletions(-)
```

```
diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid.h linux-2.6.22-rc4-mm2-2/include/linux/pid.h
--- linux-2.6.22-rc4-mm2.orig/include/linux/pid.h 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/include/linux/pid.h 2007-07-04 19:00:38.000000000 +0400
@@ -83,6 +89,9 @@ extern void FASTCALL(detach_pid(struct t
extern void FASTCALL(transfer_pid(struct task_struct *old,
    struct task_struct *new, enum pid_type));
```

```
+struct pid_namespace;
+extern struct pid_namespace init_pid_ns;
+
+/*
+ * look up a PID in the hash table. Must be called with the tasklist_lock
+ * or rcu_read_lock() held.
@@ -93,14 +99,36 @@ extern void FASTCALL(detach_pid(struct t
extern struct pid *alloc_pid(void);
extern void FASTCALL(free_pid(struct pid *pid));
```

```
+/*
+ * the helpers to get the pid's id seen from different namespaces
+ *
+ * pid_nr() : global id, i.e. the id seen from the init namespace;
+ * pid_vnr() : virtual id, i.e. the id seen from the namespace this pid
+ * belongs to. this only makes sense when called in the
+ * context of the task that belongs to the same namespace;
+ * pid_nr_ns() : id seen from the ns specified.
+ *
+ * see also task_xid_nr() etc in include/linux/sched.h
+ */
```

```
+
static inline pid_t pid_nr(struct pid *pid)
{
    pid_t nr = 0;
    if (pid)
- nr = pid->nr;
+ nr = pid->numbers[0].nr;
    return nr;
}
```

```
+pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns);
+
+static inline pid_t pid_vnr(struct pid *pid)
```



```

+{
+ pid_t nr = 0;
+ if (pid)
+ nr = pid->numbers[pid->level].nr;
+ return nr;
+}
+
#define do_each_pid_task(pid, type, task) \
do { \
    struct hlist_node *pos___; \
diff -upr linux-2.6.22-rc4-mm2.orig/kernel/pid.c linux-2.6.22-rc4-mm2-2/kernel/pid.c
--- linux-2.6.22-rc4-mm2.orig/kernel/pid.c 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/kernel/pid.c 2007-07-04 19:00:38.000000000 +0400
@@ -339,6 +379,14 @@ struct pid *find_get_pid(pid_t nr)
    return pid;
}

+pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
+{
+ pid_t nr = 0;
+ if (pid && ns->level <= pid->level)
+ nr = pid->numbers[ns->level].nr;
+ return nr;
+}
+
/*
 * Used by proc to find the first pid that is greater then or equal to nr.
 *
diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/sched.h
linux-2.6.22-rc4-mm2-2/include/linux/sched.h
--- linux-2.6.22-rc4-mm2.orig/include/linux/sched.h 2007-07-04 19:00:38.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/include/linux/sched.h 2007-07-04 19:00:38.000000000 +0400
@@ -1153,16 +1154,6 @@ struct task_struct {
    #endif
};

-static inline pid_t task_pgrp_nr(struct task_struct *tsk)
-{
- return tsk->signal->pgrp;
-}
-
-static inline pid_t task_session_nr(struct task_struct *tsk)
-{
- return tsk->signal->__session;
-}
-
static inline void set_task_session(struct task_struct *tsk, pid_t session)
{

```

```

    tsk->signal->__session = session;
@@ -1188,6 +1179,104 @@ static inline struct pid *task_session(s
    return task->group_leader->pids[PIDTYPE_SID].pid;
}

```

```

+struct pid_namespace;
+
+/*
+ * the helpers to get the task's different pids as they are seen
+ * from various namespaces
+ *
+ * task_xid_nr()    : global id, i.e. the id seen from the init namespace;
+ * task_xid_vnr()  : virtual id, i.e. the id seen from the namespace the task
+ *                  belongs to. this only makes sense when called in the
+ *                  context of the task that belongs to the same namespace;
+ * task_xid_nr_ns() : id seen from the ns specified;
+ *
+ * set_task_vxid() : assigns a virtual id to a task;
+ *
+ * task_ppid_nr_ns() : the parent's id as seen from the namespace specified.
+ *                   the result depends on the namespace and whether the
+ *                   task in question is the namespace's init. e.g. for the
+ *                   namespace's init this will return 0 when called from
+ *                   the namespace of this init, or appropriate id otherwise.
+ *
+ *
+ * see also pid_nr() etc in include/linux/pid.h
+ */
+
+static inline pid_t task_pid_nr(struct task_struct *tsk)
+{
+    return tsk->pid;
+}
+
+static inline pid_t task_pid_nr_ns(struct task_struct *tsk,
+    struct pid_namespace *ns)
+{
+    return pid_nr_ns(task_pid(tsk), ns);
+}
+
+static inline pid_t task_pid_vnr(struct task_struct *tsk)
+{
+    return pid_vnr(task_pid(tsk));
+}
+
+static inline pid_t task_tgid_nr(struct task_struct *tsk)
+{

```

```

+ return tsk->tgid;
+}
+
+static inline pid_t task_tgid_nr_ns(struct task_struct *tsk,
+ struct pid_namespace *ns)
+{
+ return pid_nr_ns(task_tgid(tsk), ns);
+}
+
+
+static inline pid_t task_tgid_vnr(struct task_struct *tsk)
+{
+ return pid_vnr(task_tgid(tsk));
+}
+
+
+
+static inline pid_t task_pgrp_nr(struct task_struct *tsk)
+{
+ return tsk->signal->pgrp;
+}
+
+
+static inline pid_t task_pgrp_nr_ns(struct task_struct *tsk,
+ struct pid_namespace *ns)
+{
+ return pid_nr_ns(task_pgrp(tsk), ns);
+}
+
+
+static inline pid_t task_pgrp_vnr(struct task_struct *tsk)
+{
+ return pid_vnr(task_pgrp(tsk));
+}
+
+
+
+static inline pid_t task_session_nr(struct task_struct *tsk)
+{
+ return tsk->signal->__session;
+}
+
+
+static inline pid_t task_session_nr_ns(struct task_struct *tsk,
+ struct pid_namespace *ns)
+{
+ return pid_nr_ns(task_session(tsk), ns);
+}
+
+
+static inline pid_t task_session_vnr(struct task_struct *tsk)
+{
+ return pid_vnr(task_session(tsk));
+}
+
+

```

```

+
+static inline pid_t task_ppid_nr_ns(struct task_struct *tsk,
+ struct pid_namespace *ns)
+{
+ return pid_nr_ns(task_pid(rcu_dereference(tsk->real_parent)), ns);
+}
+
+/**
+ * pid_alive - check that a task structure is not stale
+ * @p: Task structure to be checked.

```

---

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

---



---

Subject: [PATCH 7/16] Helpers to find the task by its numerical ids  
Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:07:02 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

When searching the task by numerical id one may need to find it using global pid (as it is done now in kernel) or by its virtual id, e.g. when sending a signal to a task from one namespace the sender will specify the task's virtual id.

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

---

```

fs/proc/base.c      |  2 +-
include/linux/pid.h | 13 ++++++++
include/linux/sched.h | 31 ++++++++
kernel/pid.c        | 32 ++++++++
4 files changed, 58 insertions(+), 20 deletions(-)

```

```

--- ./fs/proc/base.c.ve6 2007-07-06 10:58:56.000000000 +0400
+++ ./fs/proc/base.c 2007-07-06 11:03:41.000000000 +0400
@@ -2230,7 +2230,7 @@ static struct task_struct *next_tgid(uns
    rcu_read_lock());
retry:
    task = NULL;
- pid = find_ge_pid(tgid);
+ pid = find_ge_pid(tgid, &init_pid_ns);
    if (pid) {
        tgid = pid->nr + 1;
        task = pid_task(pid, PIDTYPE_PID);
--- ./include/linux/pid.h.ve6 2007-07-06 11:03:27.000000000 +0400
+++ ./include/linux/pid.h 2007-07-06 11:03:27.000000000 +0400

```

```

@@ -98,14 +98,23 @@ extern struct pid_namespace init_pid_ns;
/*
 * look up a PID in the hash table. Must be called with the tasklist_lock
 * or rcu_read_lock() held.
+ *
+ * find_pid_ns() finds the pid in the namespace specified
+ * find_pid() find the pid by its global id, i.e. in the init namespace
+ * find_vpid() find the pid by its virtual id, i.e. in the current namespace
+ *
+ * see also find_task_by_pid() set in include/linux/sched.h
 */
-extern struct pid *FASTCALL(find_pid(int nr));
+extern struct pid *FASTCALL(find_pid_ns(int nr, struct pid_namespace *ns));
+
+#define find_vpid(pid) find_pid_ns(pid, current->nsproxy->pid_ns)
+#define find_pid(pid) find_pid_ns(pid, &init_pid_ns)

/*
 * Lookup a PID in the hash table, and return with it's count elevated.
 */
extern struct pid *find_get_pid(int nr);
-extern struct pid *find_ge_pid(int nr);
+extern struct pid *find_ge_pid(int nr, struct pid_namespace *);

extern struct pid *alloc_pid(void);
extern void FASTCALL(free_pid(struct pid *pid));
--- ./include/linux/sched.h.ve6 2007-07-06 11:03:27.000000000 +0400
+++ ./include/linux/sched.h 2007-07-06 11:03:27.000000000 +0400
@@ -1475,8 +1475,35 @@ extern struct task_struct init_task;

extern struct mm_struct init_mm;

-#define find_task_by_pid(nr) find_task_by_pid_type(PIDTYPE_PID, nr)
-extern struct task_struct *find_task_by_pid_type(int type, int pid);
+extern struct pid_namespace init_pid_ns;
+
+/*
+ * find a task by one of its numerical ids
+ *
+ * find_task_by_pid_type_ns():
+ *     it is the most generic call - it finds a task by all id,
+ *     type and namespace specified
+ * find_task_by_pid_ns():
+ *     finds a task by its pid in the specified namespace
+ * find_task_by_pid_type():
+ *     finds a task by its global id with the specified type, e.g.
+ *     by global session id
+ * find_task_by_pid():

```

```

+ *   finds a task by its global pid
+ *
+ * see also find_pid() etc in include/linux/pid.h
+ */
+
+extern struct task_struct *find_task_by_pid_type_ns(int type, int pid,
+ struct pid_namespace *ns);
+
+#define find_task_by_pid_ns(nr, ns) \
+ find_task_by_pid_type_ns(PIDTYPE_PID, nr, ns)
+#define find_task_by_pid_type(type, nr) \
+ find_task_by_pid_type_ns(type, nr, &init_pid_ns)
+#define find_task_by_pid(nr) \
+ find_task_by_pid_type(PIDTYPE_PID, nr)
+
+extern void __set_special_pids(pid_t session, pid_t pgrp);

/* per-UID process charging. */
--- ./kernel/pid.c.ve6 2007-07-06 11:03:27.000000000 +0400
+++ ./kernel/pid.c 2007-07-06 11:03:27.000000000 +0400
@@ -238,19 +238,20 @@ out_free:
    goto out;
}

-struct pid * fastcall find_pid(int nr)
+struct pid * fastcall find_pid_ns(int nr, struct pid_namespace *ns)
{
    struct hlist_node *elem;
- struct pid *pid;
+ struct pid_number *pnr;
+
+ hlist_for_each_entry_rcu(pnr, elem,
+ &pid_hash[pid_hashfn(nr)], pid_chain)
+ if (pnr->nr == nr && pnr->ns == ns)
+ return container_of(pnr, struct pid,
+ numbers[ns->level]);

- hlist_for_each_entry_rcu(pid, elem,
- &pid_hash[pid_hashfn(nr)], pid_chain) {
- if (pid->nr == nr)
- return pid;
- }
return NULL;
}
-EXPORT_SYMBOL_GPL(find_pid);
+EXPORT_SYMBOL_GPL(find_pid_ns);

/*

```

```

* attach_pid() must be called with the tasklist_lock write-held.
@@ -310,12 +311,13 @@ struct task_struct * fastcall pid_task(s
/*
* Must be called under rcu_read_lock() or with tasklist_lock read-held.
*/
-struct task_struct *find_task_by_pid_type(int type, int nr)
+struct task_struct *find_task_by_pid_type_ns(int type, int nr,
+ struct pid_namespace *ns)
{
- return pid_task(find_pid(nr), type);
+ return pid_task(find_pid_ns(nr, ns), type);
}

-EXPORT_SYMBOL(find_task_by_pid_type);
+EXPORT_SYMBOL(find_task_by_pid_type_ns);

struct pid *get_task_pid(struct task_struct *task, enum pid_type type)
{
@@ -342,7 +344,7 @@ struct pid *find_get_pid(pid_t nr)
struct pid *pid;

rcu_read_lock();
- pid = get_pid(find_pid(nr));
+ pid = get_pid(find_vpid(nr));
rcu_read_unlock();

return pid;
@@ -361,15 +363,15 @@ pid_t pid_nr_ns(struct pid *pid, struct
*
* If there is a pid at nr this function is exactly the same as find_pid.
*/
-struct pid *find_ge_pid(int nr)
+struct pid *find_ge_pid(int nr, struct pid_namespace *ns)
{
struct pid *pid;

do {
- pid = find_pid(nr);
+ pid = find_pid_ns(nr, ns);
if (pid)
break;
- nr = next_pidmap(current->nsproxy->pid_ns, nr);
+ nr = next_pidmap(ns, nr);
} while (nr > 0);

return pid;

```

---

Containers mailing list

---

Subject: [PATCH 8/16] Masquerade the siginfo when sending a pid to a foreign namespace

Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:07:37 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

When user send signal from (say) init namespace to any task in a sub namespace the siginfo struct must not carry the sender's pid value, as this value may refer to some task in the destination namespace and thus may confuse the application.

The consensus was to pretend in this case as if it is the kernel who sends the signal.

The pid\_ns\_accessible() call is introduced to check this pid-to-ns accessibility.

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

---

```
include/linux/pid.h | 10 ++++++++
kernel/signal.c    | 34 ++++++-----
2 files changed, 38 insertions(+), 6 deletions(-)
```

```
diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid.h linux-2.6.22-rc4-mm2-2/include/linux/pid.h
--- linux-2.6.22-rc4-mm2.orig/include/linux/pid.h 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/include/linux/pid.h 2007-07-04 19:00:38.000000000 +0400
@@ -83,6 +89,16 @@ extern void FASTCALL(detach_pid(struct t
     return nr;
 }

+/*
+ * checks whether the pid actually lives in the namespace ns, i.e. it was
+ * created in this namespace or it was moved there.
+ */
+
+static inline int pid_ns_accessible(struct pid_namespace *ns, struct pid *pid)
+{
+ return pid->numbers[pid->level].ns == ns;
+}
+
+#define do_each_pid_task(pid, type, task) \
do { \
    struct hlist_node *pos___; \
```



```
diff -upr linux-2.6.22-rc4-mm2.orig/kernel/signal.c linux-2.6.22-rc4-mm2-2/kernel/signal.c
--- linux-2.6.22-rc4-mm2.orig/kernel/signal.c 2007-07-04 19:00:38.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/kernel/signal.c 2007-07-04 19:00:38.000000000 +0400
@@ -1124,13 +1124,31 @@ EXPORT_SYMBOL_GPL(kill_pid_info_as_uid);
 * is probably wrong. Should make it like BSD or SYSV.
 */
```

```
-static int kill_something_info(int sig, struct siginfo *info, int pid)
+static inline void masquerade_siginfo(struct pid_namespace *src_ns,
+ struct pid *tgt_pid, struct siginfo *info)
+{
+ if (tgt_pid != NULL && !pid_ns_accessible(src_ns, tgt_pid)) {
+ /*
+ * current namespace is not seen from the taks we
+ * want to send the signal to, so pretend as if it
+ * is the kernel who does this to avoid pid messing
+ * by the target
+ */
+
+ info->si_pid = 0;
+ info->si_code = SI_KERNEL;
+ }
+}
```

```
+static int kill_something_info(int sig, struct siginfo *info, int pid_nr)
{
int ret;
+ struct pid *pid;
+
rcu_read_lock();
- if (!pid) {
+ if (!pid_nr) {
ret = kill_pgrp_info(sig, info, task_pgrp(current));
- } else if (pid == -1) {
+ } else if (pid_nr == -1) {
int retval = 0, count = 0;
struct task_struct * p;
```

```
@@ -1145,10 +1163,14 @@ static int kill_something_info(int sig,
}
read_unlock(&tasklist_lock);
ret = count ? retval : -ESRCH;
- } else if (pid < 0) {
- ret = kill_pgrp_info(sig, info, find_pid(-pid));
+ } else if (pid_nr < 0) {
+ pid = find_vpid(-pid_nr);
+ masquerade_siginfo(current->nsproxy->pid_ns, pid, info);
+ ret = kill_pgrp_info(sig, info, pid);
```

```

} else {
- ret = kill_pid_info(sig, info, find_pid(pid));
+ pid = find_vpid(pid_nr);
+ masquerade_siginfo(current->nsproxy->pid_ns, pid, info);
+ ret = kill_pid_info(sig, info, pid);
}
rcu_read_unlock();
return ret;

```

---

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

---



---

Subject: [PATCH 9/16] Make proc\_flust\_task to flush entries from multiple proc trees

Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:08:09 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Since a task will appear in more than one proc tree we need to shrink many trees. For this case we pass the struct pid to proc\_flush\_task() and shrink the mounts of all the namespaces this pid belongs to.

The NULL passed to it means that only global mount is to be flushed.

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

---

```

fs/proc/base.c      | 25 ++++++-----
include/linux/proc_fs.h | 6 +++++-
kernel/exit.c       | 18 ++++++-----
3 files changed, 43 insertions(+), 6 deletions(-)

```

```
diff -upr linux-2.6.22-rc4-mm2.orig/fs/proc/base.c linux-2.6.22-rc4-mm2-2/fs/proc/base.c
```

```
--- linux-2.6.22-rc4-mm2.orig/fs/proc/base.c 2007-06-14 12:14:29.000000000 +0400
```

```
+++ linux-2.6.22-rc4-mm2-2/fs/proc/base.c 2007-07-04 19:00:38.000000000 +0400
```

```
@@ -75,6 +75,7 @@
```

```
#include <linux/nsproxy.h>
```

```
#include <linux/oom.h>
```

```
#include <linux/elf.h>
```

```
+#include <linux/pid_namespace.h>
```

```
#include "internal.h"
```

```
/* NOTE:
```

```
@@ -2183,7 +2184,7 @@ static const struct inode_operations pro
```

```
* that no dcache entries will exist at process exit time it
```

```

*      just makes it very unlikely that any will persist.
*/
-void proc_flush_task(struct task_struct *task)
+static void proc_flush_task_mnt(struct task_struct *task, struct vfsmount *mnt)
{
    struct dentry *dentry, *leader, *dir;
    char buf[PROC_NUMBUF];
@@ -2191,7 +2192,7 @@ void proc_flush_task(struct task_struct

    name.name = buf;
    name.len = snprintf(buf, sizeof(buf), "%d", task->pid);
- dentry = d_hash_and_lookup(proc_mnt->mnt_root, &name);
+ dentry = d_hash_and_lookup(mnt->mnt_root, &name);
    if (dentry) {
        shrink_dcache_parent(dentry);
        d_drop(dentry);
@@ -2203,7 +2204,7 @@ void proc_flush_task(struct task_struct

    name.name = buf;
    name.len = snprintf(buf, sizeof(buf), "%d", task->tgid);
- leader = d_hash_and_lookup(proc_mnt->mnt_root, &name);
+ leader = d_hash_and_lookup(mnt->mnt_root, &name);
    if (!leader)
        goto out;

@@ -2229,6 +2230,24 @@ out:
    return;
}

+/*
+ * when flushing dentries from proc one need to flush them from global
+ * proc (proc_mnt) and from all the namespaces' procs this task was seen
+ * in. this call is supposed to make all this job.
+ */
+
+void proc_flush_task(struct task_struct *task, struct pid *pid)
+{
+ int i;
+
+ proc_flush_task_mnt(task, proc_mnt);
+ if (pid == NULL)
+ return;
+
+ for (i = 1; i <= pid->level; i++)
+ proc_flush_task_mnt(task, pid->numbers[i].ns->proc_mnt);
+}
+
static struct dentry *proc_pid_instantiate(struct inode *dir,

```

```

    struct dentry * dentry,
    struct task_struct *task, const void *ptr)
diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/proc_fs.h
linux-2.6.22-rc4-mm2-2/include/linux/proc_fs.h
--- linux-2.6.22-rc4-mm2.orig/include/linux/proc_fs.h 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/include/linux/proc_fs.h 2007-07-04 19:00:38.000000000 +0400
@@ -111,7 +111,7 @@ extern void proc_misc_init(void);

struct mm_struct;

-void proc_flush_task(struct task_struct *task);
+void proc_flush_task(struct task_struct *task, struct pid *pid);
struct dentry *proc_pid_lookup(struct inode *dir, struct dentry * dentry, struct nameidata *);
int proc_pid_readdir(struct file * filp, void * dirent, filldir_t filldir);
unsigned long task_vsize(struct mm_struct *);
@@ -223,7 +227,9 @@ static inline void proc_net_remove(const
#define proc_net_create(name, mode, info) ({ (void)(mode), NULL; })
static inline void proc_net_remove(const char *name) {}

-static inline void proc_flush_task(struct task_struct *task) { }
+static inline void proc_flush_task(struct task_struct *task, struct pid *pid)
+{
+}

static inline struct proc_dir_entry *create_proc_entry(const char *name,
mode_t mode, struct proc_dir_entry *parent) { return NULL; }
diff -upr linux-2.6.22-rc4-mm2.orig/kernel/exit.c linux-2.6.22-rc4-mm2-2/kernel/exit.c
--- linux-2.6.22-rc4-mm2.orig/kernel/exit.c 2007-07-04 19:00:38.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/kernel/exit.c 2007-07-04 19:00:38.000000000 +0400
@@ -154,6 +154,7 @@ static void delayed_put_task_struct(stru

void release_task(struct task_struct * p)
{
+ struct pid *pid;
struct task_struct *leader;
int zap_leader;
repeat:
@@ -161,6 +162,20 @@ repeat:
write_lock_irq(&tasklist_lock);
ptrace_unlink(p);
BUG_ON(!list_empty(&p->ptrace_list) || !list_empty(&p->ptrace_children));
+ /*
+ * we have to keep this pid till proc_flush_task() to make
+ * it possible to flush all dentries holding it. pid will
+ * be put ibidem
+ *
+ * however if the pid belongs to init namespace only, we can
+ * optimize this out

```

```

+ */
+ pid = task_pid(p);
+ if (!pid_ns_accessible(&init_pid_ns, pid))
+ get_pid(pid);
+ else
+ pid = NULL;
+
+ __exit_signal(p);

/*
@@ -185,7 +200,8 @@ repeat:
}

write_unlock_irq(&tasklist_lock);
- proc_flush_task(p);
+ proc_flush_task(p, pid);
+ put_pid(pid);
release_thread(p);
call_rcu(&p->rcu, delayed_put_task_struct);

```

---

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

---



---

Subject: [PATCH 10/16] Changes in copy\_process() to work with pid namespaces  
Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:08:40 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

We must pass the namespace pointer to the alloc\_pid() to show what namespace to allocate the pid from and we should call this *after* the namespace is copied.

Essentially, the task->pid etc initialization is done after the alloc\_pid().

To do so I move the alloc\_pid() inside copy\_process() and introduce an argument to the alloc\_pid() function.

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

---

```

include/linux/pid.h | 2 +-
kernel/fork.c      | 29 ++++++-----
kernel/pid.c       | 2 +-

```

3 files changed, 19 insertions(+), 14 deletions(-)

```
--- ./include/linux/pid.h.ve9 2007-07-06 11:03:55.000000000 +0400
+++ ./include/linux/pid.h 2007-07-06 11:03:55.000000000 +0400
@@ -116,7 +116,7 @@ extern struct pid *FASTCALL(find_pid_ns(
extern struct pid *find_get_pid(int nr);
extern struct pid *find_ge_pid(int nr, struct pid_namespace *);

-extern struct pid *alloc_pid(void);
+extern struct pid *alloc_pid(struct pid_namespace *ns);
extern void FASTCALL(free_pid(struct pid *pid));

/*
--- ./kernel/fork.c.ve9 2007-07-06 11:03:55.000000000 +0400
+++ ./kernel/fork.c 2007-07-06 11:04:07.000000000 +0400
@@ -50,6 +50,7 @@
#include <linux/taskstats_kern.h>
#include <linux/random.h>
#include <linux/tty.h>
+#include <linux/pid.h>

#include <asm/pgtable.h>
#include <asm/pgalloc.h>
@@ -1032,7 +1033,6 @@ static struct task_struct *copy_process(
p->did_exec = 0;
delayacct_tsk_init(p); /* Must remain after dup_task_struct() */
copy_flags(clone_flags, p);
- p->pid = pid_nr(pid);
INIT_LIST_HEAD(&p->children);
INIT_LIST_HEAD(&p->sibling);
p->vfork_done = NULL;
@@ -1107,10 +1107,6 @@ static struct task_struct *copy_process(
p->blocked_on = NULL; /* not blocked yet */
#endif

- p->tgid = p->pid;
- if (clone_flags & CLONE_THREAD)
- p->tgid = current->tgid;
-
if ((retval = security_task_alloc(p))
goto bad_fork_cleanup_policy;
if ((retval = audit_alloc(p))
@@ -1132,9 +1128,14 @@ static struct task_struct *copy_process(
goto bad_fork_cleanup_mm;
if ((retval = copy_namespaces(clone_flags, p))
goto bad_fork_cleanup_keys;
+ if (likely(pid == NULL)) {
+ pid = alloc_pid(p->nsproxy->pid_ns);
```

```

+ if (pid == NULL)
+ goto bad_fork_cleanup_namespaces;
+ }
  retval = copy_thread(0, clone_flags, stack_start, stack_size, p, regs);
  if (retval)
- goto bad_fork_cleanup_namespaces;
+ goto bad_fork_cleanup_pid;

  p->set_child_tid = (clone_flags & CLONE_CHILD_SETTID) ? child_tidptr : NULL;
  /*
@@ -1255,6 +1256,11 @@ static struct task_struct *copy_process(
  }
  }

+ p->pid = pid_nr(pid);
+ p->tgid = p->pid;
+ if (clone_flags & CLONE_THREAD)
+ p->tgid = current->tgid;
+
  if (likely(p->pid)) {
    add_parent(p);
    if (unlikely(p->ptrace & PT_PTRACED))
@@ -1288,6 +1294,8 @@ static struct task_struct *copy_process(
  proc_fork_connector(p);
  return p;

+bad_fork_cleanup_pid:
+ free_pid(pid);
bad_fork_cleanup_namespaces:
  exit_task_namespaces(p);
bad_fork_cleanup_keys:
@@ -1380,19 +1388,16 @@ long do_fork(unsigned long clone_flags,
  {
    struct task_struct *p;
    int trace = 0;
- struct pid *pid = alloc_pid();
    long nr;

- if (!pid)
- return -EAGAIN;
- nr = pid->nr;
  if (unlikely(current->ptrace)) {
    trace = fork_traceflag (clone_flags);
    if (trace)
      clone_flags |= CLONE_PTRACE;
  }

- p = copy_process(clone_flags, stack_start, regs, stack_size, parent_tidptr, child_tidptr, pid);

```

```

+ p = copy_process(clone_flags, stack_start, regs, stack_size,
+ parent_tidptr, 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.
@@ -1418,6 +1423,7 @@ long do_fork(unsigned long clone_flags,
else
p->state = TASK_STOPPED;

+ nr = pid_vnr(task_pid(p));
if (unlikely (trace)) {
current->ptrace_message = nr;
ptrace_notify ((trace << 8) | SIGTRAP);
@@ -1433,7 +1439,6 @@ long do_fork(unsigned long clone_flags,
}
}
} else {
- free_pid(pid);
nr = PTR_ERR(p);
}
return nr;
--- ./kernel/pid.c.ve9 2007-07-06 11:03:55.000000000 +0400
+++ ./kernel/pid.c 2007-07-06 11:03:55.000000000 +0400
@@ -206,7 +206,7 @@ fastcall void free_pid(struct pid *pid)
call_rcu(&pid->rcu, delayed_put_pid);
}

-struct pid *alloc_pid(void)
+struct pid *alloc_pid(struct pid_namespace *pid_ns)
{
struct pid *pid;
enum pid_type type;

```

---

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

---



---

**Subject: [PATCH 11/16] Add support for multiple kmem caches for pids**  
Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:09:21 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Unike Suka's patches I don not limit the level of pid nesting  
creating the caches on demand, depending on the namespace's level.

Each kmem cache is names "pid\_<NR>", where <NR> is the level  
of pid namespace and thus - the number of virtual pids in it.



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

---

pid.c | 61 +++  
1 files changed, 56 insertions(+), 5 deletions(-)

```
--- ./kernel/pid.c.ve10 2007-07-06 11:04:15.000000000 +0400
+++ ./kernel/pid.c 2007-07-06 11:04:48.000000000 +0400
@@ -32,7 +32,6 @@
#define pid_hashfn(nr) hash_long((unsigned long)nr, pidhash_shift)
static struct hlist_head *pid_hash;
static int pidhash_shift;
-static struct kmem_cache *pid_cachep;
struct pid init_struct_pid = INIT_STRUCT_PID;

int pid_max = PID_MAX_DEFAULT;
@@ -179,11 +178,15 @@ static int next_pidmap(struct pid_namesp

fastcall void put_pid(struct pid *pid)
{
+ struct pid_namespace *ns;
+
  if (!pid)
    return;
+
+ ns = pid->numbers[0].ns;
  if ((atomic_read(&pid->count) == 1) ||
      atomic_dec_and_test(&pid->count))
- kmem_cache_free(pid_cachep, pid);
+ kmem_cache_free(ns->pid_cachep, pid);
}
EXPORT_SYMBOL_GPL(put_pid);

@@ -212,7 +215,7 @@ struct pid *alloc_pid(struct pid_namespa
enum pid_type type;
int nr = -1;

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

@@ -233,7 +236,7 @@ out:
return pid;

out_free:
- kmem_cache_free(pid_cachep, pid);
```

```

+ kmem_cache_free(init_pid_ns.pid_cachep, pid);
  pid = NULL;
  goto out;
}
@@ -378,6 +381,52 @@ struct pid *find_ge_pid(int nr, struct p
}
EXPORT_SYMBOL_GPL(find_get_pid);

+struct pid_cache {
+ int level;
+ char name[16];
+ struct kmem_cache *cachep;
+ struct list_head lh;
+};
+
+static LIST_HEAD(pid_caches);
+static DEFINE_MUTEX(pid_cache_mutex);
+
+static struct kmem_cache *create_pid_cachep(int level)
+{
+ struct pid_cache *pc;
+ struct kmem_cache *cachep = NULL;
+
+ mutex_lock(&pid_cache_mutex);
+ list_for_each_entry (pc, &pid_caches, lh)
+ if (pc->level == level) {
+   cachep = pc->cachep;
+   goto out;
+ }
+
+ pc = kzalloc(sizeof(struct pid_cache), GFP_KERNEL);
+ if (pc == NULL)
+   goto out;
+
+ snprintf(pc->name, sizeof(pc->name), "pid_%d", level);
+ cachep = kmem_cache_create(pc->name,
+   sizeof(struct pid) + level * sizeof(struct pid_number),
+   0, SLAB_HWCACHE_ALIGN, NULL, NULL);
+ if (cachep == NULL)
+   goto out_free;
+
+ pc->cachep = cachep;
+ pc->level = level;
+ list_add(&pc->lh, &pid_caches);
+ pc = NULL;
+
+out_free:
+ if (pc != NULL)

```

```

+ kfree(pc);
+out:
+ mutex_unlock(&pid_cache_mutex);
+ return cachep;
+}
+
struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *old_ns)
{
    BUG_ON(!old_ns);
@@ -425,5 +474,7 @@ void __init pidmap_init(void)
    set_bit(0, init_pid_ns.pidmap[0].page);
    atomic_dec(&init_pid_ns.pidmap[0].nr_free);

- pid_cachep = KMEM_CACHE(pid, SLAB_PANIC);
+ init_pid_ns.pid_cachep = create_pid_cachep(0);
+ if (init_pid_ns.pid_cachep == NULL)
+ panic("Can't create pid cachep");
}

```

---

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

---



---

Subject: [PATCH 12/16] Reference counting of pid namespaces by pids  
Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:10:06 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Getting and putting the pid namespace in alloc\_pid() and free\_pid() is too slow. Instead this I get/put the namespace by the pidmaps. When the pidmap allocates its first pid the namespace is get, when the pidmap becomes empty - the namespace is put.

Although pids may live longer than their "fingerprints" in the pidmaps, this is ok to release the namespace with not yet freed struct pids, as this pid is not alive and no routines will use it.

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

---

pid.c | 19 ++++++++-----  
1 files changed, 15 insertions(+), 4 deletions(-)

```

diff -upr linux-2.6.22-rc4-mm2.orig/kernel/pid.c linux-2.6.22-rc4-mm2-2/kernel/pid.c
--- linux-2.6.22-rc4-mm2.orig/kernel/pid.c 2007-06-14 12:14:29.000000000 +0400

```

```
+++ linux-2.6.22-rc4-mm2-2/kernel/pid.c 2007-07-04 19:00:38.000000000 +0400
@@ -89,16 +95,23 @@ static __cacheline_aligned_in_smp DEFIN
```

```
static fastcall void free_pidmap(struct pid_namespace *pid_ns, int pid)
{
- struct pidmap *map = pid_ns->pidmap + pid / BITS_PER_PAGE;
  int offset = pid & BITS_PER_PAGE_MASK;
+ int map_id = pid / BITS_PER_PAGE;
+ struct pidmap *map = pid_ns->pidmap + map_id;
+ int free_pids;

  clear_bit(offset, map->page);
- atomic_inc(&map->nr_free);
+ free_pids = atomic_inc_return(&map->nr_free);
+
+ if (map_id == 0)
+   free_pids++;
+ if (free_pids == BITS_PER_PAGE)
+   put_pid_ns(pid_ns);
}
```

```
static int alloc_pidmap(struct pid_namespace *pid_ns)
{
- int i, offset, max_scan, pid, last = pid_ns->last_pid;
+ int i, offset, max_scan, pid, last = pid_ns->last_pid, free_pids;
  struct pidmap *map;
```

```
  pid = last + 1;
@@ -126,7 +139,11 @@ static int alloc_pidmap(struct pid_names
  if (likely(atomic_read(&map->nr_free))) {
    do {
      if (!test_and_set_bit(offset, map->page)) {
-       atomic_dec(&map->nr_free);
+       free_pids = atomic_dec_return(
+         &map->nr_free);
+       if (free_pids == BITS_PER_PAGE - 1)
+         get_pid_ns(pid_ns);
+
        pid_ns->last_pid = pid;
        return pid;
      }
}
```

---

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

---

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_namespa
{
  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>

---



---

Subject: [PATCH 14/16] Make pid namespaces clonnable  
Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:11:18 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Just add the support for cloning pid namespaces.

Note that the namespace is destroyed via `schedule_work()`. This is done so, since the namespace will put the `proc_mnt` and thus may fall asleep.

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

---

```

include/linux/pid_namespace.h | 5 +-
include/linux/sched.h         | 1
kernel/fork.c                 | 26 ++++++-----
kernel/nsproxy.c              | 2
kernel/pid.c                  | 96 ++++++-----
5 files changed, 118 insertions(+), 12 deletions(-)

```

```

--- ./include/linux/pid_namespace.h.ve13 2007-07-06 11:05:05.000000000 +0400
+++ ./include/linux/pid_namespace.h 2007-07-06 11:05:05.000000000 +0400
@@ -15,7 +15,10 @@ 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;
int level;
--- ./include/linux/sched.h.ve13 2007-07-06 11:05:05.000000000 +0400
+++ ./include/linux/sched.h 2007-07-06 11:05:23.000000000 +0400
@@ -26,6 +26,7 @@
#define CLONE_NEWUTS 0x04000000 /* New utsname group? */
#define CLONE_NEWIPC 0x08000000 /* New ipcs */
#define CLONE_NEWUSER 0x10000000 /* New user namespace */
+#define CLONE_NEWPIDS 0x20000000 /* New pids */

/*
 * Scheduling policies
--- ./kernel/fork.c.ve13 2007-07-06 11:05:05.000000000 +0400
+++ ./kernel/fork.c 2007-07-06 11:05:05.000000000 +0400
@@ -1267,11 +1267,22 @@ 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_NEWPIDS) {
+ 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)++;
@@ -1423,7 +1434,10 @@ long do_fork(unsigned long clone_flags,
else
p->state = TASK_STOPPED;

- nr = pid_vnr(task_pid(p));
+ nr = (clone_flags & CLONE_NEWPIDS) ?
+ pid_nr_ns(task_pid(p), current->nsproxy->pid_ns) :
+ pid_vnr(task_pid(p));
+
if (unlikely (trace)) {
current->ptrace_message = nr;
ptrace_notify ((trace << 8) | SIGTRAP);
--- ./kernel/nsproxy.c.ve13 2007-07-06 10:58:57.000000000 +0400
+++ ./kernel/nsproxy.c 2007-07-06 11:05:39.000000000 +0400
@@ -184,7 +184,7 @@ int unshare_nsproxy_namespaces(unsigned
int err = 0;

if (!(unshare_flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
- CLONE_NEWUSER)))
+ CLONE_NEWUSER | CLONE_NEWPIDS)))
return 0;

if (!capable(CAP_SYS_ADMIN))
--- ./kernel/pid.c.ve13 2007-07-06 11:05:05.000000000 +0400
+++ ./kernel/pid.c 2007-07-06 11:06:47.000000000 +0400
@@ -62,8 +62,10 @@ 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 }
@@ -457,11 +459,96 @@ out:
return cachep;
}

```

```

+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);
+ if (ns->pid_cachep == NULL)
+ goto out_free_map;
+
+ kref_init(&ns->kref);
+ ns->last_pid = 0;
+ ns->child_reaper = NULL;
+ ns->level = level;
+
+ if (pid_ns_prepare_proc(ns))
+ goto out_free_cachep;
+
+ 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_cachep:
+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;
+
+ synchronize_rcu();
+ pid_ns_release_proc(ns);
+ atomic_inc(&ns->pidmap[0].nr_free);
+ 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);
+ get_pid_ns(old_ns);
- return old_ns;
+ new_ns = old_ns;
+ if (!(flags & CLONE_NEWPIDS))
+ 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)
@@ -469,7 +556,8 @@ 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);
}

/*

```

---

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

---



---

Subject: [PATCH 15/16] Changes to show virtual ids to user  
Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:13:20 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

This is the largest patch in the set. Make all (I hope) the places where the pid is shown to or get from user operate on the virtual pids.

The idea is:

- all in-kernel data structures must store either struct pid itself or the pid's global nr, obtained with pid\_nr() call;
- when seeking the task from kernel code with the stored id one should use find\_task\_by\_pid() call that works with global pids;
- when showing pid's numerical value to the user the virtual one should be used, but however when one shows task's pid outside this task's namespace the global one is to be used;
- when getting the pid from userspace one need to consider this as the virtual one and use appropriate task/pid-searching functions.

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

---

```

arch/ia64/kernel/signal.c | 4 +--
arch/parisc/kernel/signal.c | 2 +-
drivers/char/tty_io.c | 7 +++++-
fs/binfmt_elf.c | 16 ++++++-----
fs/binfmt_elf_fdpic.c | 16 ++++++-----
fs/exec.c | 4 +--
fs/proc/array.c | 21 ++++++-----
fs/proc/base.c | 37 ++++++-----
include/net/scm.h | 4 +--
ipc/mqueue.c | 4 +--
ipc/msg.c | 6 +---

```

```

ipc/sem.c          |  8 ++++----
ipc/shm.c          |  6 ++++---
kernel/capability.c | 13 ++++++++-----
kernel/exit.c      | 31 ++++++++-----
kernel/fork.c      |  2 +-
kernel/futex.c     | 20 ++++++++-----
kernel/ptrace.c    |  4 +++-
kernel/sched.c     |  3 +-
kernel/signal.c    | 26 ++++++++-----
kernel/sys.c       | 38 ++++++++-----
kernel/timer.c     |  7 ++++---
net/core/scm.c     |  4 +++-
net/unix/af_unix.c |  6 ++++---
24 files changed, 173 insertions(+), 116 deletions(-)

```

```

--- ./arch/ia64/kernel/signal.c.ve14 2007-05-29 13:34:03.000000000 +0400
+++ ./arch/ia64/kernel/signal.c 2007-07-06 11:07:04.000000000 +0400
@@ -227,7 +227,7 @@ ia64_rt_sigreturn (struct sigscratch *sc
     si.si_signo = SIGSEGV;
     si.si_errno = 0;
     si.si_code = SI_KERNEL;
- si.si_pid = current->pid;
+ si.si_pid = task_pid_vnr(current);
     si.si_uid = current->uid;
     si.si_addr = sc;
     force_sig_info(SIGSEGV, &si, current);
@@ -332,7 +332,7 @@ force_sigsegv_info (int sig, void __user
     si.si_signo = SIGSEGV;
     si.si_errno = 0;
     si.si_code = SI_KERNEL;
- si.si_pid = current->pid;
+ si.si_pid = task_pid_vnr(current);
     si.si_uid = current->uid;
     si.si_addr = addr;
     force_sig_info(SIGSEGV, &si, current);
--- ./arch/parisc/kernel/signal.c.ve14 2007-05-29 13:34:04.000000000 +0400
+++ ./arch/parisc/kernel/signal.c 2007-07-06 11:07:04.000000000 +0400
@@ -181,7 +181,7 @@ give_sigsegv:
     si.si_signo = SIGSEGV;
     si.si_errno = 0;
     si.si_code = SI_KERNEL;
- si.si_pid = current->pid;
+ si.si_pid = task_pid_vnr(current);
     si.si_uid = current->uid;
     si.si_addr = &frame->uc;
     force_sig_info(SIGSEGV, &si, current);
--- ./drivers/char/tty_io.c.ve14 2007-07-06 11:07:04.000000000 +0400
+++ ./drivers/char/tty_io.c 2007-07-06 11:07:04.000000000 +0400

```

```

@@ -103,6 +103,7 @@
#include <linux/selection.h>

#include <linux/kmod.h>
+#include <linux/nsproxy.h>

#undef TTY_DEBUG_HANGUP

@@ -3080,7 +3081,7 @@ static int tiocgpgrp(struct tty_struct *
*/
if (tty == real_tty && current->signal->tty != real_tty)
return -ENOTTY;
- return put_user(pid_nr(real_tty->pgrp), p);
+ return put_user(pid_vnr(real_tty->pgrp), p);
}

/**
@@ -3114,7 +3115,7 @@ static int tiocspgrp(struct tty_struct *
if (pgrp_nr < 0)
return -EINVAL;
rcu_read_lock();
- pgrp = find_pid(pgrp_nr);
+ pgrp = find_vpid(pgrp_nr);
retval = -ESRCH;
if (!pgrp)
goto out_unlock;
@@ -3151,7 +3152,7 @@ static int tiocgsid(struct tty_struct *t
return -ENOTTY;
if (!real_tty->session)
return -ENOTTY;
- return put_user(pid_nr(real_tty->session), p);
+ return put_user(pid_vnr(real_tty->session), p);
}

/**
--- ./fs/binfmt_elf.c.ve14 2007-07-06 11:07:04.000000000 +0400
+++ ./fs/binfmt_elf.c 2007-07-06 11:07:04.000000000 +0400
@@ -1402,10 +1402,10 @@ static void fill_prstatus(struct elf_prs
prstatus->pr_info.si_signo = prstatus->pr_cursig = signr;
prstatus->pr_sigpend = p->pending.signal.sig[0];
prstatus->pr_sighold = p->blocked.sig[0];
- prstatus->pr_pid = p->pid;
- prstatus->pr_ppid = p->parent->pid;
- prstatus->pr_pgrp = task_pgrp_nr(p);
- prstatus->pr_sid = task_session_nr(p);
+ prstatus->pr_pid = task_pid_vnr(p);
+ prstatus->pr_ppid = task_pid_vnr(p->parent);
+ prstatus->pr_pgrp = task_pgrp_vnr(p);

```

```

+ prstatus->pr_sid = task_session_vnr(p);
  if (thread_group_leader(p)) {
    /*
     * This is the record for the group leader. Add in the
@@ -1448,10 +1448,10 @@ static int fill_psinfo(struct elf_prpsin
  psinfo->pr_psargs[i] = ' ';
  psinfo->pr_psargs[len] = 0;

- psinfo->pr_pid = p->pid;
- psinfo->pr_ppid = p->parent->pid;
- psinfo->pr_pgrp = task_pgrp_nr(p);
- psinfo->pr_sid = task_session_nr(p);
+ psinfo->pr_pid = task_pid_vnr(p);
+ psinfo->pr_ppid = task_pid_vnr(p->parent);
+ psinfo->pr_pgrp = task_pgrp_vnr(p);
+ psinfo->pr_sid = task_session_vnr(p);

  i = p->state ? ffz(~p->state) + 1 : 0;
  psinfo->pr_state = i;
--- ./fs/binfmt_elf_fdpic.c.ve14 2007-07-06 11:07:04.000000000 +0400
+++ ./fs/binfmt_elf_fdpic.c 2007-07-06 11:07:04.000000000 +0400
@@ -1342,10 +1342,10 @@ static void fill_prstatus(struct elf_prs
  prstatus->pr_info.si_signo = prstatus->pr_cursig = signr;
  prstatus->pr_sigpend = p->pending.signal.sig[0];
  prstatus->pr_sighold = p->blocked.sig[0];
- prstatus->pr_pid = p->pid;
- prstatus->pr_ppid = p->parent->pid;
- prstatus->pr_pgrp = task_pgrp_nr(p);
- prstatus->pr_sid = task_session_nr(p);
+ prstatus->pr_pid = task_pid_vnr(p);
+ prstatus->pr_ppid = task_pid_vnr(p->parent);
+ prstatus->pr_pgrp = task_pgrp_vnr(p);
+ prstatus->pr_sid = task_session_vnr(p);
  if (thread_group_leader(p)) {
    /*
     * This is the record for the group leader. Add in the
@@ -1391,10 +1391,10 @@ static int fill_psinfo(struct elf_prpsin
  psinfo->pr_psargs[i] = ' ';
  psinfo->pr_psargs[len] = 0;

- psinfo->pr_pid = p->pid;
- psinfo->pr_ppid = p->parent->pid;
- psinfo->pr_pgrp = task_pgrp_nr(p);
- psinfo->pr_sid = task_session_nr(p);
+ psinfo->pr_pid = task_pid_vnr(p);
+ psinfo->pr_ppid = task_pid_vnr(p->parent);
+ psinfo->pr_pgrp = task_pgrp_vnr(p);
+ psinfo->pr_sid = task_session_vnr(p);

```

```

i = p->state ? ffz(~p->state) + 1 : 0;
psinfo->pr_state = i;
--- ./fs/exec.c.ve14 2007-07-06 10:58:55.000000000 +0400
+++ ./fs/exec.c 2007-07-06 11:07:04.000000000 +0400
@@ -1486,7 +1486,7 @@ static int format_corename(char *corenam
    case 'p':
        pid_in_pattern = 1;
        rc = snprintf(out_ptr, out_end - out_ptr,
-           "%d", current->tgid);
+           "%d", task_tgid_vnr(current));
        if (rc > out_end - out_ptr)
            goto out;
        out_ptr += rc;
@@ -1558,7 +1558,7 @@ static int format_corename(char *corenam
    if (!lispie && !pid_in_pattern
        && (core_uses_pid || atomic_read(&current->mm->mm_users) != 1)) {
        rc = snprintf(out_ptr, out_end - out_ptr,
-           ".%d", current->tgid);
+           ".%d", task_tgid_vnr(current));
        if (rc > out_end - out_ptr)
            goto out;
        out_ptr += rc;
--- ./fs/proc/array.c.ve14 2007-07-06 11:07:04.000000000 +0400
+++ ./fs/proc/array.c 2007-07-06 11:07:51.000000000 +0400
@@ -75,6 +75,7 @@
#include <linux/cpuset.h>
#include <linux/rcupdate.h>
#include <linux/delayacct.h>
+#include <linux/pid_namespace.h>

#include <asm/uaccess.h>
#include <asm/pgtable.h>
@@ -161,7 +162,9 @@ static inline char * task_state(struct t
    struct group_info *group_info;
    int g;
    struct fdtable *fdt = NULL;
+ struct pid_namespace *ns;

+ ns = current->nsproxy->pid_ns;
    rcu_read_lock();
    buffer += sprintf(buffer,
        "State:\t%s\n"
@@ -172,9 +175,12 @@ static inline char * task_state(struct t
        "Uid:\t%d\t%d\t%d\t%d\n"
        "Gid:\t%d\t%d\t%d\t%d\n",
    get_task_state(p),
-    p->tgid, p->pid,

```



```

- pid_alive(p) ? rcu_dereference(p->real_parent)->tgid : 0,
- pid_alive(p) && p->ptrace ? rcu_dereference(p->parent)->pid : 0,
+ task_tgid_nr_ns(p, ns),
+ task_pid_nr_ns(p, ns),
+ pid_alive(p) ?
+ task_ppid_nr_ns(p, ns) : 0,
+ pid_alive(p) && p->ptrace ?
+ task_tgid_nr_ns(rcu_dereference(p->parent), ns) : 0,
  p->uid, p->euid, p->suid, p->fsuid,
  p->gid, p->egid, p->sgid, p->fsgid);

```

```

@@ -396,6 +402,7 @@ static int do_task_stat(struct task_stru
  rcu_read_lock();
  if (lock_task_sighand(task, &flags)) {
    struct signal_struct *sig = task->signal;
+ struct pid_namespace *ns = current->nsproxy->pid_ns;

```

```

    if (sig->tty) {
      tty_pgrp = pid_nr(sig->tty->pgrp);
@@ -428,9 +435,9 @@ static int do_task_stat(struct task_stru
      stime += cputime_to_clock_t(sig->stime);
    }

```

```

- sid = task_session_nr(task);
- pgid = task_pgrp_nr(task);
- ppid = rcu_dereference(task->real_parent)->tgid;
+ sid = task_session_nr_ns(task, ns);
+ pgid = task_pgrp_nr_ns(task, ns);
+ ppid = task_ppid_nr_ns(task, ns);

```

```

  unlock_task_sighand(task, &flags);
}
@@ -461,7 +468,7 @@ static int do_task_stat(struct task_stru
  res = sprintf(buffer, "%d (%s) %c %d %d %d %d %d %u %lu \
%lu %lu %lu %lu %lu %ld %ld %ld %ld %ld 0 %llu %lu %ld %lu %lu %lu %lu \
%lu %lu %lu %lu %lu %lu %lu %lu %d %d %u %u %llu\n",

```

```

- task->pid,
+ task_pid_nr_ns(task, current->nsproxy->pid_ns),
  tcomm,
  state,
  ppid,
--- ./fs/proc/base.c.ve14 2007-07-06 11:07:04.000000000 +0400
+++ ./fs/proc/base.c 2007-07-06 11:07:04.000000000 +0400

```

```

@@ -1842,14 +1842,14 @@ static int proc_self_readlink(struct den
  int buflen)
{
  char tmp[PROC_NUMBUF];
- sprintf(tmp, "%d", current->tgid);

```

```

+ sprintf(tmp, "%d", task_tgid_vnr(current));
  return vfs_readlink(dentry,buffer,buflen,tmp);
}

static void *proc_self_follow_link(struct dentry *dentry, struct nameidata *nd)
{
  char tmp[PROC_NUMBUF];
- sprintf(tmp, "%d", current->tgid);
+ sprintf(tmp, "%d", task_tgid_vnr(current));
  return ERR_PTR(vfs_follow_link(nd,tmp));
}

@@ -2214,6 +2214,7 @@ struct dentry *proc_pid_lookup(struct in
  struct dentry *result = ERR_PTR(-ENOENT);
  struct task_struct *task;
  unsigned tgid;
+ struct pid_namespace *ns;

  result = proc_base_lookup(dir, dentry);
  if (!IS_ERR(result) || PTR_ERR(result) != -ENOENT)
@@ -2223,8 +2224,9 @@ struct dentry *proc_pid_lookup(struct in
  if (tgid == ~0U)
    goto out;

+ ns = (struct pid_namespace *)dentry->d_sb->s_fs_info;
  rcu_read_lock();
- task = find_task_by_pid(tgid);
+ task = find_task_by_pid_ns(tgid, ns);
  if (task)
    get_task_struct(task);
  rcu_read_unlock();
@@ -2241,7 +2243,8 @@ out:
  * Find the first task with tgid >= tgid
  *
  */
-static struct task_struct *next_tgid(unsigned int tgid)
+static struct task_struct *next_tgid(unsigned int tgid,
+ struct pid_namespace *ns)
{
  struct task_struct *task;
  struct pid *pid;
@@ -2249,9 +2252,9 @@ static struct task_struct *next_tgid(uns
  rcu_read_lock();
  retry:
  task = NULL;
- pid = find_ge_pid(tgid, &init_pid_ns);
+ pid = find_ge_pid(tgid, ns);
  if (pid) {

```

```

- tgid = pid->nr + 1;
+ tgid = pid_nr_ns(pid, ns) + 1;
  task = pid_task(pid, PIDTYPE_PID);
  /* What we to know is if the pid we have find is the
   * pid of a thread_group_leader. Testing for task
@@ -2291,6 +2294,7 @@ int proc_pid_readdir(struct file * filp,
  struct task_struct *reaper = get_proc_task(filp->f_path.dentry->d_inode);
  struct task_struct *task;
  int tgid;
+ struct pid_namespace *ns;

  if (!reaper)
    goto out_no_task;
@@ -2301,11 +2305,12 @@ int proc_pid_readdir(struct file * filp,
  goto out;
}

+ ns = (struct pid_namespace *)filp->f_dentry->d_sb->s_fs_info;
  tgid = filp->f_pos - TGID_OFFSET;
- for (task = next_tgid(tgid);
+ for (task = next_tgid(tgid, ns);
  task;
-  put_task_struct(task), task = next_tgid(tgid + 1)) {
- tgid = task->pid;
+  put_task_struct(task), task = next_tgid(tgid + 1, ns)) {
+ tgid = task_pid_nr_ns(task, ns);
  filp->f_pos = tgid + TGID_OFFSET;
  if (proc_pid_fill_cache(filp, dirent, filldir, task, tgid) < 0) {
    put_task_struct(task);
@@ -2434,6 +2439,7 @@ static struct dentry *proc_task_lookup(s
  struct task_struct *task;
  struct task_struct *leader = get_proc_task(dir);
  unsigned tid;
+ struct pid_namespace *ns;

  if (!leader)
    goto out_no_task;
@@ -2442,8 +2448,9 @@ static struct dentry *proc_task_lookup(s
  if (tid == ~0U)
    goto out;

+ ns = (struct pid_namespace *)dentry->d_sb->s_fs_info;
  rcu_read_lock();
- task = find_task_by_pid(tid);
+ task = find_task_by_pid_ns(tid, ns);
  if (task)
    get_task_struct(task);
  rcu_read_unlock();

```

```

@@ -2474,14 +2481,14 @@ out_no_task:
 * threads past it.
 */
static struct task_struct *first_tid(struct task_struct *leader,
- int tid, int nr)
+ int tid, int nr, struct pid_namespace *ns)
{
    struct task_struct *pos;

    rcu_read_lock();
    /* Attempt to start with the pid of a thread */
    if (tid && (nr > 0)) {
- pos = find_task_by_pid(tid);
+ pos = find_task_by_pid_ns(tid, ns);
    if (pos && (pos->group_leader == leader))
        goto found;
    }
@@ -2550,6 +2557,7 @@ static int proc_task_readdir(struct file
    ino_t ino;
    int tid;
    unsigned long pos = filp->f_pos; /* avoiding "long long" filp->f_pos */
+ struct pid_namespace *ns;

    task = get_proc_task(inode);
    if (!task)
@@ -2583,12 +2591,13 @@ static int proc_task_readdir(struct file
    /* f_version caches the tgid value that the last readdir call couldn't
     * return. lseek aka telldir automagically resets f_version to 0.
     */
+ ns = (struct pid_namespace *)filp->f_dentry->d_sb->s_fs_info;
    tid = filp->f_version;
    filp->f_version = 0;
- for (task = first_tid(leader, tid, pos - 2);
+ for (task = first_tid(leader, tid, pos - 2, ns);
        task;
        task = next_tid(task), pos++) {
- tid = task->pid;
+ tid = task_pid_nr_ns(task, ns);
    if (proc_task_fill_cache(filp, dirent, filldir, task, tid) < 0) {
        /* returning this tgid failed, save it as the first
         * pid for the next readir call */
--- ./include/net/scm.h.ve14 2007-05-18 19:44:43.000000000 +0400
+++ ./include/net/scm.h 2007-07-06 11:07:04.000000000 +0400
@@ -4,6 +4,8 @@
#include <linux/limits.h>
#include <linux/net.h>
#include <linux/security.h>
+#include <linux/pid.h>

```

```

+#include <linux/nsproxy.h>

/* Well, we should have at least one descriptor open
 * to accept passed FDs 8)
@@ -54,7 +56,7 @@ static __inline__ int scm_send(struct so
    struct task_struct *p = current;
    scm->creds.uid = p->uid;
    scm->creds.gid = p->gid;
- scm->creds.pid = p->tgid;
+ scm->creds.pid = task_tgid_vnr(p);
    scm->fp = NULL;
    scm->seq = 0;
    unix_get_peersec_dgram(sock, scm);
--- ./ipc/mqueue.c.ve14 2007-07-06 10:36:20.000000000 +0400
+++ ./ipc/mqueue.c 2007-07-06 11:07:04.000000000 +0400
@@ -29,6 +29,8 @@
#include <linux/audit.h>
#include <linux/signal.h>
#include <linux/mutex.h>
+#include <linux/nsproxy.h>
+#include <linux/pid.h>

#include <net/sock.h>
#include "util.h"
@@ -513,7 +515,7 @@ static void __do_notify(struct mqueue_in
    sig_i.si_errno = 0;
    sig_i.si_code = SI_MESGQ;
    sig_i.si_value = info->notify.sigev_value;
- sig_i.si_pid = current->tgid;
+ sig_i.si_pid = task_pid_vnr(current);
    sig_i.si_uid = current->uid;

    kill_pid_info(info->notify.sigev_signo,
--- ./ipc/msg.c.ve14 2007-07-06 10:58:57.000000000 +0400
+++ ./ipc/msg.c 2007-07-06 11:07:04.000000000 +0400
@@ -613,7 +613,7 @@ static inline int pipelined_send(struct
    msr->r_msg = ERR_PTR(-E2BIG);
    } else {
    msr->r_msg = NULL;
- msq->q_lrp_id = msr->r_tsk->pid;
+ msq->q_lrp_id = task_pid_vnr(msr->r_tsk);
    msq->q_rtime = get_seconds();
    wake_up_process(msr->r_tsk);
    smp_mb();
@@ -697,7 +697,7 @@ long do_msgsnd(int msqid, long mtype, vo
    }
}

```

```

- msq->q_lspid = current->tgid;
+ msq->q_lspid = task_tgid_vnr(current);
  msq->q_stime = get_seconds();

  if (!pipelined_send(msq, msg)) {
@@ -812,7 +812,7 @@ long do_msgrcv(int msqid, long *pmtyp,
  list_del(&msg->m_list);
  msq->q_qnum--;
  msq->q_rtime = get_seconds();
- msq->q_lrpipid = current->tgid;
+ msq->q_lrpipid = task_tgid_vnr(current);
  msq->q_cbytes -= msg->m_ts;
  atomic_sub(msg->m_ts, &msg_bytes);
  atomic_dec(&msg_hdrs);
--- ./ipc/sem.c.ve14 2007-07-06 10:58:57.000000000 +0400
+++ ./ipc/sem.c 2007-07-06 11:07:04.000000000 +0400
@@ -795,7 +795,7 @@ static int semctl_main(struct ipc_namesp
  for (un = sma->undo; un; un = un->id_next)
    un->semadj[semnum] = 0;
  curr->semval = val;
- curr->sempid = current->tgid;
+ curr->sempid = task_tgid_vnr(current);
  sma->sem_ctime = get_seconds();
  /* maybe some queued-up processes were waiting for this */
  update_queue(sma);
@@ -1198,7 +1198,7 @@ retry_undos:
  if (error)
    goto out_unlock_free;

- error = try_atomic_semop (sma, sops, nsops, un, current->tgid);
+ error = try_atomic_semop (sma, sops, nsops, un, task_tgid_vnr(current));
  if (error <= 0) {
    if (alter && error == 0)
      update_queue (sma);
@@ -1213,7 +1213,7 @@ retry_undos:
  queue.sops = sops;
  queue.nsops = nsops;
  queue.undo = un;
- queue.pid = current->tgid;
+ queue.pid = task_tgid_vnr(current);
  queue.id = semid;
  queue.alter = alter;
  if (alter)
@@ -1384,7 +1384,7 @@ found:
  semaphore->semval = 0;
  if (semaphore->semval > SEMVMX)
    semaphore->semval = SEMVMX;
- semaphore->sempid = current->tgid;

```

```

+ semaphore->sempid = task_tgid_vnr(current);
}
}
sma->sem_otime = get_seconds();
--- ./ipc/shm.c.ve14 2007-07-06 10:58:57.000000000 +0400
+++ ./ipc/shm.c 2007-07-06 11:07:04.000000000 +0400
@@ -168,7 +168,7 @@ static void shm_open(struct vm_area_stru
shp = shm_lock(sfd->ns, sfd->id);
BUG_ON(!shp);
shp->shm_atim = get_seconds();
- shp->shm_lprid = current->tgid;
+ shp->shm_lprid = task_tgid_vnr(current);
shp->shm_nattch++;
shm_unlock(shp);
}
@@ -213,7 +213,7 @@ static void shm_close(struct vm_area_str
/* remove from the list of attaches of the shm segment */
shp = shm_lock(ns, sfd->id);
BUG_ON(!shp);
- shp->shm_lprid = current->tgid;
+ shp->shm_lprid = task_tgid_vnr(current);
shp->shm_dtim = get_seconds();
shp->shm_nattch--;
if(shp->shm_nattch == 0 &&
@@ -390,7 +390,7 @@ static int newseg (struct ipc_namespace
if(id == -1)
goto no_id;

- shp->shm_cprid = current->tgid;
+ shp->shm_cprid = task_tgid_vnr(current);
shp->shm_lprid = 0;
shp->shm_atim = shp->shm_dtim = 0;
shp->shm_ctim = get_seconds();
--- ./kernel/capability.c.ve14 2007-07-06 10:36:20.000000000 +0400
+++ ./kernel/capability.c 2007-07-06 11:07:04.000000000 +0400
@@ -12,6 +12,7 @@
#include <linux/module.h>
#include <linux/security.h>
#include <linux/syscalls.h>
+#include <linux/pid_namespace.h>
#include <asm/uaccess.h>

unsigned securebits = SECUREBITS_DEFAULT; /* systemwide security settings */
@@ -67,8 +68,9 @@ asmlinkage long sys_capget(cap_user_head
spin_lock(&task_capability_lock);
read_lock(&tasklist_lock);

- if (pid && pid != current->pid) {

```

```

- target = find_task_by_pid(pid);
+ if (pid && pid != task_pid_vnr(current)) {
+   target = find_task_by_pid_ns(pid,
+     current->nsproxy->pid_ns);
  if (!target) {
    ret = -ESRCH;
    goto out;
@@ -190,7 +192,7 @@ asmlinkage long sys_capset(cap_user_head
  if (get_user(pid, &header->pid))
    return -EFAULT;

- if (pid && pid != current->pid && !capable(CAP_SETPCAP))
+ if (pid && pid != task_pid_vnr(current) && !capable(CAP_SETPCAP))
    return -EPERM;

  if (copy_from_user(&effective, &data->effective, sizeof(effective)) ||
@@ -201,8 +203,9 @@ asmlinkage long sys_capset(cap_user_head
  spin_lock(&task_capability_lock);
  read_lock(&tasklist_lock);

- if (pid > 0 && pid != current->pid) {
-   target = find_task_by_pid(pid);
+ if (pid > 0 && pid != task_pid_vnr(current)) {
+   target = find_task_by_pid_ns(pid,
+     current->nsproxy->pid_ns);
  if (!target) {
    ret = -ESRCH;
    goto out;
--- ./kernel/exit.c.ve14 2007-07-06 11:07:04.000000000 +0400
+++ ./kernel/exit.c 2007-07-06 11:07:04.000000000 +0400
@@ -1120,15 +1120,17 @@ asmlinkage void sys_exit_group(int error
static int eligible_child(pid_t pid, int options, struct task_struct *p)
{
  int err;
+ struct pid_namespace *ns;

+ ns = current->nsproxy->pid_ns;
  if (pid > 0) {
- if (p->pid != pid)
+ if (task_pid_nr_ns(p, ns) != pid)
    return 0;
  } else if (!pid) {
- if (task_pgrp_nr(p) != task_pgrp_nr(current))
+ if (task_pgrp_nr_ns(p, ns) != task_pgrp_vnr(current))
    return 0;
  } else if (pid != -1) {
- if (task_pgrp_nr(p) != -pid)
+ if (task_pgrp_nr_ns(p, ns) != -pid)

```



```

    return 0;
}

@@ -1199,9 +1201,12 @@ static int wait_task_zombie(struct task_
    unsigned long state;
    int retval;
    int status;
+ struct pid_namespace *ns;
+
+ ns = current->nsproxy->pid_ns;

    if (unlikely(noreap)) {
- pid_t pid = p->pid;
+ pid_t pid = task_pid_nr_ns(p, ns);
    uid_t uid = p->uid;
    int exit_code = p->exit_code;
    int why, status;
@@ -1319,7 +1324,7 @@ static int wait_task_zombie(struct task_
    retval = put_user(status, &infop->si_status);
}
if (!retval && infop)
- retval = put_user(p->pid, &infop->si_pid);
+ retval = put_user(task_pid_nr_ns(p, ns), &infop->si_pid);
if (!retval && infop)
    retval = put_user(p->uid, &infop->si_uid);
if (retval) {
@@ -1327,7 +1332,7 @@ static int wait_task_zombie(struct task_
    p->exit_state = EXIT_ZOMBIE;
    return retval;
}
- retval = p->pid;
+ retval = task_pid_nr_ns(p, ns);
if (p->real_parent != p->parent) {
    write_lock_irq(&tasklist_lock);
    /* Double-check with lock held. */
@@ -1365,6 +1370,7 @@ static int wait_task_stopped(struct task
    int __user *stat_addr, struct rusage __user *ru)
{
    int retval, exit_code;
+ struct pid_namespace *ns;

    if (!p->exit_code)
        return 0;
@@ -1383,11 +1389,12 @@ static int wait_task_stopped(struct task
    * keep holding onto the tasklist_lock while we call getrusage and
    * possibly take page faults for user memory.
    */
+ ns = current->nsproxy->pid_ns;

```

```

get_task_struct(p);
read_unlock(&tasklist_lock);

if (unlikely(noreap)) {
- pid_t pid = p->pid;
+ pid_t pid = task_pid_nr_ns(p, ns);
  uid_t uid = p->uid;
  int why = (p->ptrace & PT_PTRACED) ? CLD_TRAPPED : CLD_STOPPED;

```

```

@@ -1458,11 +1465,11 @@ bail_ref:
  if (!retval && infop)
    retval = put_user(exit_code, &infop->si_status);
  if (!retval && infop)
-  retval = put_user(p->pid, &infop->si_pid);
+  retval = put_user(task_pid_nr_ns(p, ns), &infop->si_pid);
  if (!retval && infop)
    retval = put_user(p->uid, &infop->si_uid);
  if (!retval)
-  retval = p->pid;
+  retval = task_pid_nr_ns(p, ns);
  put_task_struct(p);

```

```

BUG_ON(!retval);
@@ -1482,6 +1489,7 @@ static int wait_task_continued(struct ta
  int retval;
  pid_t pid;
  uid_t uid;
+ struct pid_namespace *ns;

```

```

  if (unlikely(!p->signal))
    return 0;
@@ -1499,7 +1507,8 @@ static int wait_task_continued(struct ta
  p->signal->flags &= ~SIGNAL_STOP_CONTINUED;
  spin_unlock_irq(&p->sigband->siglock);

```

```

- pid = p->pid;
+ ns = current->nsproxy->pid_ns;
+ pid = task_pid_nr_ns(p, ns);
  uid = p->uid;
  get_task_struct(p);
  read_unlock(&tasklist_lock);
@@ -1510,7 +1519,7 @@ static int wait_task_continued(struct ta
  if (!retval && stat_addr)
    retval = put_user(0xffff, stat_addr);
  if (!retval)
-  retval = p->pid;
+  retval = task_pid_nr_ns(p, ns);
  } else {

```

```

    retval = wait_noreap_copyout(p, pid, uid,
        CLD_CONTINUED, SIGCONT,
--- ./kernel/fork.c.ve14 2007-07-06 11:07:04.000000000 +0400
+++ ./kernel/fork.c 2007-07-06 11:07:04.000000000 +0400
@@ -1300,7 +1300,7 @@ static struct task_struct *copy_process(
    * TID. It's too late to back out if this fails.
    */
    if (clone_flags & CLONE_PARENT_SETTID)
- put_user(p->pid, parent_tidptr);
+ put_user(task_pid_vnr(p), parent_tidptr);

    proc_fork_connector(p);
    return p;
--- ./kernel/futex.c.ve14 2007-07-06 10:58:57.000000000 +0400
+++ ./kernel/futex.c 2007-07-06 11:09:13.000000000 +0400
@@ -52,6 +52,8 @@
#include <linux/syscalls.h>
#include <linux/signal.h>
#include <linux/module.h>
+#include <linux/pid.h>
+#include <linux/nsproxy.h>
#include <asm/futex.h>

#include "rtmutex_common.h"
@@ -647,7 +649,7 @@ static int wake_futex_pi(u32 __user *uad
    if (!(uval & FUTEX_OWNER_DIED)) {
        int ret = 0;

- newval = FUTEX_WAITERS | new_owner->pid;
+ newval = FUTEX_WAITERS | task_pid_vnr(new_owner);

        curval = cmpxchg_futex_value_locked(uaddr, uval, newval);

@@ -1100,7 +1102,7 @@ static void unqueue_me_pi(struct futex_q
    static int fixup_pi_state_owner(u32 __user *uaddr, struct futex_q *q,
        struct task_struct *curr)
    {
- u32 newtid = curr->pid | FUTEX_WAITERS;
+ u32 newtid = task_pid_vnr(curr) | FUTEX_WAITERS;
        struct futex_pi_state *pi_state = q->pi_state;
        u32 uval, curval, newval;
        int ret;
@@ -1362,7 +1364,7 @@ static int futex_lock_pi(u32 __user *uad
    * (by doing a 0 -> TID atomic cmpxchg), while holding all
    * the locks. It will most likely not succeed.
    */
- newval = current->pid;
+ newval = task_pid_vnr(current);

```

```

curval = cmpxchg_futex_value_locked(uaddr, 0, newval);

@@ -1373,7 +1375,7 @@ static int futex_lock_pi(u32 __user *uad
 * Detect deadlocks. In case of REQUEUE_PI this is a valid
 * situation and we return success to user space.
 */
- if (unlikely((curval & FUTEX_TID_MASK) == current->pid)) {
+ if (unlikely((curval & FUTEX_TID_MASK) == task_pid_vnr(current))) {
    ret = -EDEADLK;
    goto out_unlock_release_sem;
}
@@ -1402,7 +1404,7 @@ static int futex_lock_pi(u32 __user *uad
 */
if (unlikely(ownerdied || !(curval & FUTEX_TID_MASK))) {
    /* Keep the OWNER_DIED bit */
- newval = (curval & ~FUTEX_TID_MASK) | current->pid;
+ newval = (curval & ~FUTEX_TID_MASK) | task_pid_vnr(current);
    ownerdied = 0;
    lock_taken = 1;
}
@@ -1581,7 +1583,7 @@ retry:
/*
 * We release only a lock we actually own:
 */
- if ((uval & FUTEX_TID_MASK) != current->pid)
+ if ((uval & FUTEX_TID_MASK) != task_pid_vnr(current))
    return -EPERM;
/*
 * First take all the futex related locks:
@@ -1602,7 +1604,7 @@ retry_unlocked:
 * anyone else up:
 */
if (!(uval & FUTEX_OWNER_DIED))
- uval = cmpxchg_futex_value_locked(uaddr, current->pid, 0);
+ uval = cmpxchg_futex_value_locked(uaddr, task_pid_vnr(current), 0);

if (unlikely(uval == -EFAULT))
@@ -1611,7 +1613,7 @@ retry_unlocked:
 * Rare case: we managed to release the lock atomically,
 * no need to wake anyone else up:
 */
- if (unlikely(uval == current->pid))
+ if (unlikely(uval == task_pid_vnr(current)))
    goto out_unlock;

/*

```

```

@@ -1880,7 +1882,7 @@ retry:
    if (get_user(uval, uaddr))
        return -1;

- if ((uval & FUTEX_TID_MASK) == curr->pid) {
+ if ((uval & FUTEX_TID_MASK) == task_pid_vnr(curr)) {
    /*
     * Ok, this dying thread is truly holding a futex
     * of interest. Set the OWNER_DIED bit atomically
--- ./kernel/ptrace.c.ve14 2007-07-06 10:58:57.000000000 +0400
+++ ./kernel/ptrace.c 2007-07-06 11:07:04.000000000 +0400
@@ -19,6 +19,7 @@
#include <linux/security.h>
#include <linux/signal.h>
#include <linux/audit.h>
+#include <linux/pid_namespace.h>

#include <asm/pgtable.h>
#include <asm/uaccess.h>
@@ -439,7 +440,8 @@ struct task_struct *ptrace_get_task_stru
    return ERR_PTR(-EPERM);

    read_lock(&tasklist_lock);
- child = find_task_by_pid(pid);
+ child = find_task_by_pid_ns(pid,
+ current->nsproxy->pid_ns);
    if (child)
        get_task_struct(child);

--- ./kernel/sched.c.ve14 2007-07-06 10:58:57.000000000 +0400
+++ ./kernel/sched.c 2007-07-06 11:07:04.000000000 +0400
@@ -61,6 +61,7 @@
#include <linux/delayacct.h>
#include <linux/reciprocal_div.h>
#include <linux/cpu_acct.h>
+#include <linux/pid_namespace.h>

#include <asm/tlb.h>
#include <asm/unistd.h>
@@ -1662,7 +1663,7 @@ asmlinkage void schedule_tail(struct tas
    preempt_enable();
#endif
    if (current->set_child_tid)
- put_user(current->pid, current->set_child_tid);
+ put_user(task_pid_vnr(current), current->set_child_tid);
}

/*

```

```

--- ./kernel/signal.c.ve14 2007-07-06 11:07:04.000000000 +0400
+++ ./kernel/signal.c 2007-07-06 11:07:04.000000000 +0400
@@ -697,7 +697,7 @@ static int send_signal(int sig, struct s
    q->info.si_signo = sig;
    q->info.si_errno = 0;
    q->info.si_code = SI_USER;
-   q->info.si_pid = current->pid;
+   q->info.si_pid = task_pid_vnr(current);
    q->info.si_uid = current->uid;
    break;
    case (unsigned long) SEND_SIG_PRIV:
@@ -1475,7 +1475,11 @@ void do_notify_parent(struct task_struct

    info.si_signo = sig;
    info.si_errno = 0;
-   info.si_pid = tsk->pid;
+   /*
+    * we are under tasklist_lock here so our parent is tied to
+    * us and cannot exit and release its namespace.
+    */
+   info.si_pid = task_pid_nr_ns(tsk, tsk->parent->nsproxy->pid_ns);
    info.si_uid = tsk->uid;

    /* FIXME: find out whether or not this is supposed to be c*time. */
@@ -1540,7 +1544,11 @@ static void do_notify_parent_cldstop(str

    info.si_signo = SIGCHLD;
    info.si_errno = 0;
-   info.si_pid = tsk->pid;
+   /*
+    * we are under tasklist_lock here so our parent is tied to
+    * us and cannot exit and release its namespace.
+    */
+   info.si_pid = task_pid_nr_ns(tsk, tsk->parent->nsproxy->pid_ns);
    info.si_uid = tsk->uid;

    /* FIXME: find out whether or not this is supposed to be c*time. */
@@ -1670,7 +1678,7 @@ void ptrace_notify(int exit_code)
    memset(&info, 0, sizeof info);
    info.si_signo = SIGTRAP;
    info.si_code = exit_code;
-   info.si_pid = current->pid;
+   info.si_pid = task_pid_vnr(current);
    info.si_uid = current->uid;

    /* Let the debugger run. */
@@ -1840,7 +1848,7 @@ relock:
    info->si_signo = signr;

```

```

    info->si_errno = 0;
    info->si_code = SI_USER;
-   info->si_pid = current->parent->pid;
+   info->si_pid = task_pid_vnr(current->parent);
    info->si_uid = current->parent->uid;
}

@@ -2229,7 +2237,7 @@ sys_kill(int pid, int sig)
    info.si_signo = sig;
    info.si_errno = 0;
    info.si_code = SI_USER;
-   info.si_pid = current->tgid;
+   info.si_pid = task_tgid_vnr(current);
    info.si_uid = current->uid;

    return kill_something_info(sig, &info, pid);
@@ -2245,12 +2253,12 @@ static int do_tkill(int tgid, int pid, i
    info.si_signo = sig;
    info.si_errno = 0;
    info.si_code = SI_TKILL;
-   info.si_pid = current->tgid;
+   info.si_pid = task_tgid_vnr(current);
    info.si_uid = current->uid;

    read_lock(&tasklist_lock);
-   p = find_task_by_pid(pid);
-   if (p && (tgid <= 0 || p->tgid == tgid)) {
+   p = find_task_by_pid_ns(pid, current->nsproxy->pid_ns);
+   if (p && (tgid <= 0 || task_tgid_vnr(p) == tgid)) {
        error = check_kill_permission(sig, &info, p);
/*
 * The null signal is a permissions and process existence
--- ./kernel/sys.c.ve14 2007-07-06 11:07:04.000000000 +0400
+++ ./kernel/sys.c 2007-07-06 11:07:04.000000000 +0400
@@ -676,7 +676,8 @@ asmlinkage long sys_setpriority(int whic
    switch (which) {
        case PRIO_PROCESS:
            if (who)
-               p = find_task_by_pid(who);
+               p = find_task_by_pid_ns(who,
+               current->nsproxy->pid_ns);
            else
                p = current;
            if (p)
@@ -733,7 +734,8 @@ asmlinkage long sys_getpriority(int whic
    switch (which) {
        case PRIO_PROCESS:
            if (who)

```

```

- p = find_task_by_pid(who);
+ p = find_task_by_pid_ns(who,
+   current->nsproxy->pid_ns);
  else
    p = current;
    if (p) {
@@ -1438,7 +1440,7 @@ asmlinkage long sys_setpgid(pid_t pid, p
  int err = -EINVAL;

  if (!pid)
- pid = group_leader->pid;
+ pid = task_pid_vnr(group_leader);
  if (!pgid)
    pgid = pid;
  if (pgid < 0)
@@ -1450,7 +1452,7 @@ asmlinkage long sys_setpgid(pid_t pid, p
  write_lock_irq(&tasklist_lock);

  err = -ESRCH;
- p = find_task_by_pid(pid);
+ p = find_task_by_pid_ns(pid, current->nsproxy->pid_ns);
  if (!p)
    goto out;

@@ -1477,7 +1479,8 @@ asmlinkage long sys_setpgid(pid_t pid, p

  if (pgid != pid) {
    struct task_struct *g =
- find_task_by_pid_type(PIDTYPE_PGID, pgid);
+ find_task_by_pid_type_ns(PIDTYPE_PGID, pgid,
+   current->nsproxy->pid_ns);

    if (!g || task_session(g) != task_session(group_leader))
      goto out;
@@ -1488,9 +1491,12 @@ asmlinkage long sys_setpgid(pid_t pid, p
    goto out;

    if (task_pgrp_nr(p) != pgid) {
+ struct pid *pid;
+
    detach_pid(p, PIDTYPE_PGID);
- p->signal->pgrp = pgid;
- attach_pid(p, PIDTYPE_PGID, find_pid(pgid));
+ pid = find_vpid(pgid);
+ attach_pid(p, PIDTYPE_PGID, pid);
+ p->signal->pgrp = pid_nr(pid);
    }

```



```

err = 0;
@@ -1503,19 +1509,20 @@ out:
asmlinkage long sys_getpgid(pid_t pid)
{
if (!pid)
- return task_pgrp_nr(current);
+ return task_pgrp_vnr(current);
else {
int retval;
struct task_struct *p;

read_lock(&tasklist_lock);
- p = find_task_by_pid(pid);
+ p = find_task_by_pid_ns(pid,
+ current->nsproxy->pid_ns);

retval = -ESRCH;
if (p) {
retval = security_task_getpgid(p);
if (!retval)
- retval = task_pgrp_nr(p);
+ retval = task_pgrp_vnr(p);
}
read_unlock(&tasklist_lock);
return retval;
@@ -1527,7 +1534,7 @@ asmlinkage long sys_getpgid(pid_t pid)
asmlinkage long sys_getpgrp(void)
{
/* SMP - assuming writes are word atomic this is fine */
- return task_pgrp_nr(current);
+ return task_pgrp_vnr(current);
}

#endif
@@ -1535,19 +1542,20 @@ asmlinkage long sys_getpgrp(void)
asmlinkage long sys_getsid(pid_t pid)
{
if (!pid)
- return task_session_nr(current);
+ return task_session_vnr(current);
else {
int retval;
struct task_struct *p;

read_lock(&tasklist_lock);
- p = find_task_by_pid(pid);
+ p = find_task_by_pid_ns(pid,
+ current->nsproxy->pid_ns);

```

```

    retval = -ESRCH;
    if (p) {
        retval = security_task_getsid(p);
        if (!retval)
-   retval = task_session_nr(p);
+   retval = task_session_vnr(p);
    }
    read_unlock(&tasklist_lock);
    return retval;
@@ -1584,7 +1592,7 @@ asmlinkage long sys_setsid(void)
    group_leader->signal->tty = NULL;
    spin_unlock(&group_leader->siglock);

- err = task_pgrp_nr(group_leader);
+ err = task_pgrp_vnr(group_leader);
out:
    write_unlock_irq(&tasklist_lock);
    return err;
--- ./kernel/timer.c.ve14 2007-07-06 10:58:57.000000000 +0400
+++ ./kernel/timer.c 2007-07-06 11:09:24.000000000 +0400
@@ -37,6 +37,7 @@
#include <linux/tick.h>
#include <linux/kallsyms.h>
#include <linux/kgdb.h>
+#include <linux/pid_namespace.h>

#include <asm/uaccess.h>
#include <asm/unistd.h>
@@ -957,7 +958,7 @@ asmlinkage unsigned long sys_alarm(unsig
    */
    asmlinkage long sys_getpid(void)
    {
- return current->tgid;
+ return task_tgid_vnr(current);
    }

/*
@@ -971,7 +972,7 @@ asmlinkage long sys_getppid(void)
    int pid;

    rcu_read_lock();
- pid = rcu_dereference(current->real_parent)->tgid;
+ pid = task_ppid_nr_ns(current, current->nsproxy->pid_ns);
    rcu_read_unlock();

    return pid;
@@ -1103,7 +1104,7 @@ EXPORT_SYMBOL(schedule_timeout_uninterru

```

```

/* Thread ID - the internal kernel "pid" */
asmlinkage long sys_gettid(void)
{
- return current->pid;
+ return task_pid_vnr(current);
}

/**
--- ./net/core/scm.c.ve14 2007-07-06 10:58:57.000000000 +0400
+++ ./net/core/scm.c 2007-07-06 11:07:04.000000000 +0400
@@ -24,6 +24,8 @@
#include <linux/interrupt.h>
#include <linux/netdevice.h>
#include <linux/security.h>
+#include <linux/pid.h>
+#include <linux/nsproxy.h>

#include <asm/system.h>
#include <asm/uaccess.h>
@@ -42,7 +44,7 @@

static __inline__ int scm_check_creds(struct ucred *creds)
{
- if ((creds->pid == current->tgid || capable(CAP_SYS_ADMIN)) &&
+ if ((creds->pid == task_tgid_vnr(current) || capable(CAP_SYS_ADMIN)) &&
    ((creds->uid == current->uid || creds->uid == current->euid ||
     creds->uid == current->suid) || capable(CAP_SETUID)) &&
    ((creds->gid == current->gid || creds->gid == current->egid ||
--- ./net/unix/af_unix.c.ve14 2007-07-06 10:41:41.000000000 +0400
+++ ./net/unix/af_unix.c 2007-07-06 11:07:04.000000000 +0400
@@ -456,7 +456,7 @@ static int unix_listen(struct socket *so
    sk->sk_max_ack_backlog = backlog;
    sk->sk_state = TCP_LISTEN;
    /* set credentials so connect can copy them */
- sk->sk_peercred.pid = current->tgid;
+ sk->sk_peercred.pid = task_tgid_vnr(current);
    sk->sk_peercred.uid = current->euid;
    sk->sk_peercred.gid = current->egid;
    err = 0;
@@ -1102,7 +1102,7 @@ restart:
    unix_peer(newsk) = sk;
    newsk->sk_state = TCP_ESTABLISHED;
    newsk->sk_type = sk->sk_type;
- newsk->sk_peercred.pid = current->tgid;
+ newsk->sk_peercred.pid = task_tgid_vnr(current);
    newsk->sk_peercred.uid = current->euid;
    newsk->sk_peercred.gid = current->egid;
    newu = unix_sk(newsk);

```

```
@@ -1166,7 +1166,7 @@ static int unix_socketpair(struct socket
    sock_hold(skb);
    unix_peer(ska)=skb;
    unix_peer(skb)=ska;
- ska->sk_peercred.pid = skb->sk_peercred.pid = current->tgid;
+ ska->sk_peercred.pid = skb->sk_peercred.pid = task_tgid_vnr(current);
  ska->sk_peercred.uid = skb->sk_peercred.uid = current->euid;
  ska->sk_peercred.gid = skb->sk_peercred.gid = current->egid;
```

---

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

---

---

Subject: [PATCH 16/16] Remove already unneeded memnbers from struct pid  
Posted by [Pavel Emelianov](#) on Fri, 06 Jul 2007 08:16:29 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Since we've switched from using these we may just remove them.

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

---

```
init_task.h | 3 ---
pid.h       | 3 ---
2 files changed, 6 deletions(-)
```

```
diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid.h linux-2.6.22-rc4-mm2-2/include/linux/pid.h
--- linux-2.6.22-rc4-mm2.orig/include/linux/pid.h 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/include/linux/pid.h 2007-07-04 19:00:38.000000000 +0400
@@ -40,9 +40,6 @@ enum pid_type
 struct pid
 {
     atomic_t count;
- /* Try to keep pid_chain in the same cacheline as nr for find_pid */
- int nr;
- struct hlist_node pid_chain;
 /* lists of tasks that use this pid */
     struct hlist_head tasks[PIDTYPE_MAX];
     struct rcu_head rcu;
diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/init_task.h
linux-2.6.22-rc4-mm2-2/include/linux/init_task.h
--- linux-2.6.22-rc4-mm2.orig/include/linux/init_task.h 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/include/linux/init_task.h 2007-07-04 19:00:38.000000000 +0400
@@ -91,9 +91,6 @@ extern struct group_info init_groups;
```

```
#define INIT_STRUCT_PID { \
    .count = ATOMIC_INIT(1), \
    - .nr = 0, \
    - /* Don't put this struct pid in pid_hash */ \
    - .pid_chain = { .next = NULL, .pprev = NULL }, \
    .tasks = { \
        { .first = &init_task.pids[PIDTYPE_PID].node }, \
        { .first = &init_task.pids[PIDTYPE_PGID].node }, \
    }
```

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Herbert Poetzl](#) on Mon, 09 Jul 2007 12:02:36 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Fri, Jul 06, 2007 at 12:01:59PM +0400, Pavel Emelianov wrote:

> This is "submission for inclusion" of hierarchical, not kconfig  
> configurable, zero overheaded ;) pid namespaces.  
>  
> The overall idea is the following:  
>  
> The namespace are organized as a tree - once a task is cloned  
> with CLONE\_NEWPIDS (yes, I've also switched to it :) the new  
> namespace becomes the parent's child and tasks living in the  
> parent namespace see the tasks from the new one. The numerical  
> ids are used on the kernel-user boundary, i.e. when we export  
> pid to user we show the id, that should be used to address the  
> task in question from the namespace we're exporting this id to.

how does that behave when:

- a) the parent dies and gets reaped?
- b) the 'spawned' init dies, but other tasks inside the pid space are still active?
- c) what visibility rules do apply for the various spaces (including the default host space)?

> The main difference from Suka's patches are the following:  
>  
> 0. Suka's patches change the kernel/pid.c code too heavy.  
> This set keeps the kernel code look like it was without  
> the patches. However, this is a minor issue. The major is:  
>  
> 1. Suka's approach is to remove the notion of the task's

- > numerical pid from the kernel at all. The numbers are
- > used on the kernel-user boundary or within the kernel but
- > with the namespace this nr belongs to. This results in
- > massive changes of struct's members fro int pid to struct
- > pid \*pid, task->pid becomes the virtual id and so on and
- > so forth.
- > My approach is to keep the good old logic in the kernel.
- > The task->pid is a global and unique pid, find\_pid() finds
- > the pid by its global id and so on. The virtual ids appear
- > on the user-kernel boundary only. Thus drivers and other
- > kernel code may still be unaware of pids unless they do not
- > communicate with the userspace and get/put numerical pids.

interesting ... not sure that is what kernel folks  
have in mind though (IIRC, the struct pid change was  
considered a kernel side cleanup)

- > And some more minor differences:
- >
- > 2. Suka's patches have the limit of pid namespace nesting.
- > My patches do not.
- >
- > 3. Suka assumes that pid namespace can live without proc mount
- > and tries to make the code work with pid\_ns->proc\_mnt change
- > from NULL to not-NULL from times to times.
- > My code calls the kern\_mount() at the namespace creation and
- > thus the pid\_namespace always works with proc.

shouldn't that be done by userspace instead?

- > There are some small issues that I can describe if someone is
- > interested.
- >
- > The tests like nptl perf, unixbench spawn, getpid and others
- > didn't reveal any performance degradation in init\_namespace
- > with the RHEL5 kernel .config file. I admit, that different
- > .config-s may show that patches hurt the performance, but the
- > intention was \*not\* to make the kernel work worse with popular
- > distributions.
- >
- > This set has some ways to move forward, but this is some kind
- > of a core, that do not change the init\_pid\_namespace behavior
- > (checked with LTP tests) and may require some hacking to do
- > with the namespaces only.

TIA,  
Herbert

> Patches apply to 2.6.22-rc6-mm1.

>

> Containers mailing list

> Containers@lists.linux-foundation.org

> <https://lists.linux-foundation.org/mailman/listinfo/containers>

---

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>

---

---

Subject: Re: [PATCH 0/16] Pid namespaces

Posted by [Pavel Emelianov](#) on Mon, 09 Jul 2007 13:16:17 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Herbert Poetzl wrote:

> On Fri, Jul 06, 2007 at 12:01:59PM +0400, Pavel Emelianov wrote:

>> This is "submission for inclusion" of hierarchical, not kconfig

>> configurable, zero overheaded ;) pid namespaces.

>>

>> The overall idea is the following:

>>

>> The namespace are organized as a tree - once a task is cloned

>> with CLONE\_NEWPIDS (yes, I've also switched to it :) the new

>> namespace becomes the parent's child and tasks living in the

>> parent namespace see the tasks from the new one. The numerical

>> ids are used on the kernel-user boundary, i.e. when we export

>> pid to user we show the id, that should be used to address the

>> task in question from the namespace we're exporting this id to.

>

> how does that behave when:

>

> a) the parent dies and gets reaped?

The children are re-parented to the namespace's init.

Surprised?

> b) the 'spawned' init dies, but other tasks

> inside the pid space are still active?

The init's init becomes the namespace's init.

> c) what visibility rules do apply for the

> various spaces (including the default host space)?

Each task sees tasks from its namespace and all its children namespaces. Yes, each task can see itself as well ;)

>> The main difference from Suka's patches are the following:  
>>  
>> 0. Suka's patches change the kernel/pid.c code too heavy.  
>> This set keeps the kernel code look like it was without  
>> the patches. However, this is a minor issue. The major is:  
>>  
>> 1. Suka's approach is to remove the notion of the task's  
>> numerical pid from the kernel at all. The numbers are  
>> used on the kernel-user boundary or within the kernel but  
>> with the namespace this nr belongs to. This results in  
>> massive changes of struct's members fro int pid to struct  
>> pid \*pid, task->pid becomes the virtual id and so on and  
>> so forth.  
>> My approach is to keep the good old logic in the kernel.  
>> The task->pid is a global and unique pid, find\_pid() finds  
>> the pid by its global id and so on. The virtual ids appear  
>> on the user-kernel boundary only. Thus drivers and other  
>> kernel code may still be unaware of pids unless they do not  
>> communicate with the userspace and get/put numerical pids.  
>  
> interesting ... not sure that is what kernel folks  
> have in mind though (IIRC, the struct pid change was  
> considered a kernel side cleanup)

That's why I'm sending the patches - to make "kernel folks" make a decision. Will we see some patches from VServer team?

>> And some more minor differences:  
>>  
>> 2. Suka's patches have the limit of pid namespace nesting.  
>> My patches do not.  
>>  
>> 3. Suka assumes that pid namespace can live without proc mount  
>> and tries to make the code work with pid\_ns->proc\_mnt change  
>> from NULL to not-NULL from times to times.  
>> My code calls the kern\_mount() at the namespace creation and  
>> thus the pid\_namespace always works with proc.  
>  
> shouldn't that be done by userspace instead?

It can be. But when the namespace is being created there's no any userspace in it yet.

>> There are some small issues that I can describe if someone is  
>> interested.  
>>  
>> The tests like nptl perf, unixbench spawn, getpid and others  
>> didn't reveal any performance degradation in init\_namespace



>> with the RHEL5 kernel .config file. I admit, that different  
>> .config-s may show that patches hurt the performance, but the  
>> intention was \*not\* to make the kernel work worse with popular  
>> distributions.

>>  
>> This set has some ways to move forward, but this is some kind  
>> of a core, that do not change the init\_pid\_namespace behavior  
>> (checked with LTP tests) and may require some hacking to do  
>> with the namespaces only.

>  
> TIA,  
> Herbert  
>

BTW, why did you remove Suka and Serge from Cc?

Pavel

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Badari Pulavarty](#) on Mon, 09 Jul 2007 17:46:20 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Fri, 2007-07-06 at 12:01 +0400, Pavel Emelianov wrote:  
> This is "submission for inclusion" of hierarchical, not kconfig  
> configurable, zero overheaded ;) pid namespaces.

Not able to boot my ppc64 machine with the patchset :(

Thanks,  
Badari

Unable to handle kernel paging request for data at address 0x00000000  
Faulting instruction address: 0xc000000000247ce0  
Oops: Kernel access of bad area, sig: 11 [#1]  
SMP NR\_CPUS=32 NUMA pSeries  
Modules linked in:  
NIP: c00000000247ce0 LR: c00000000107bf4 CTR: c00000000107bd0  
REGS: c0000000005fb920 TRAP: 0300 Not tainted (2.6.22-rc6-mm1)  
MSR: 800000000009032 <EE,ME,IR,DR> CR: 24000048 XER: 20000005  
DAR: 0000000000000000, DSISR: 0000000040000000  
TASK = c000000000514650[0] 'swapper' THREAD: c0000000005f8000 CPU: 0  
GPR00: c000000000bd42c c0000000005fba0 c0000000005f8f18 0000000000000000  
GPR04: 0000000000000000 c000000000645190 c00000000d025000 c00000000d024cb8

GPR08: c0000000d024d10 0000000000000000 c0000000d024cf0 0000000000000000  
GPR12: 000000000000247f c000000000514d80 0000000000000000 c00000000044e438  
GPR16: 4000000001c00000 c00000000044ce50 0000000000000000 0000000000000000  
GPR20: c0000000004f9fd0 00000000020f9fd0 0000000000000000 c0000000005bc370  
GPR24: c0000000005bc2f8 c000000000539738 0000000002000000 0000000000000000  
GPR28: c0000000d024c00 0000000000000000 c00000000054a358 c0000000d024c00  
NIP [c00000000247ce0] .kref\_get+0x0/0x28  
LR [c000000000107bf4] .proc\_set\_super+0x24/0x54  
Call Trace:  
[c0000000005fbba0] [c0000000005fbc30] 0xc0000000005fbc30 (unreliable)  
[c0000000005fbc30] [c000000000bd42c] .sget+0x34c/0x470  
[c0000000005fbd00] [c000000000107da4] .proc\_get\_sb+0xa0/0x18c  
[c0000000005fbdb0] [c000000000bdc84] .vfs\_kern\_mount+0x80/0xe8  
[c0000000005fbe50] [c0000000004ea1e4] .proc\_root\_init+0x4c/0x158  
[c0000000005fbed0] [c0000000004cb9ec] .start\_kernel+0x3c8/0x404  
[c0000000005fbf90] [c00000000008524] .start\_here\_common+0x54/0x130  
Instruction dump:  
60000000 e8440008 7c0903a6 4e800421 e8410028 38000001 38210080 7c030378  
e8010010 ebc1fff0 7c0803a6 4e800020 <80030000> 7c000034 5400d97e 0b000000  
Kernel panic - not syncing: Attempted to kill the idle task!

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Herbert Poetzl](#) on Mon, 09 Jul 2007 19:52:41 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Mon, Jul 09, 2007 at 05:16:17PM +0400, Pavel Emelianov wrote:  
> Herbert Poetzl wrote:  
> > On Fri, Jul 06, 2007 at 12:01:59PM +0400, Pavel Emelianov wrote:  
> >> This is "submission for inclusion" of hierarchical, not kconfig  
> >> configurable, zero overheaded ;) pid namespaces.  
> >>  
> >> The overall idea is the following:  
> >>  
> >> The namespace are organized as a tree - once a task is cloned  
> >> with CLONE\_NEWPIDS (yes, I've also switched to it :) the new  
> >> namespace becomes the parent's child and tasks living in the  
> >> parent namespace see the tasks from the new one. The numerical  
> >> ids are used on the kernel-user boundary, i.e. when we export  
> >> pid to user we show the id, that should be used to address the

> >> task in question from the namespace we're exporting this id to.  
> >  
> > how does that behave when:  
> >  
> > a) the parent dies and gets reaped?  
>  
> The children are re-parented to the namespace's init.  
> Surprised?  
>  
> > b) the 'spawned' init dies, but other tasks  
> > inside the pid space are still active?  
>  
> The init's init becomes the namespace's init.

so an init from the parent process is chosen here?  
or 'the init' process? or what am I missing here?

> > c) what visibility rules do apply for the  
> > various spaces (including the default host space)?  
>  
> Each task sees tasks from its namespace and all its children  
> namespaces. Yes, each task can see itself as well ;)  
>  
> >> The main difference from Suka's patches are the following:  
> >>  
> >> 0. Suka's patches change the kernel/pid.c code too heavy.  
> >> This set keeps the kernel code look like it was without  
> >> the patches. However, this is a minor issue. The major is:  
> >>  
> >> 1. Suka's approach is to remove the notion of the task's  
> >> numerical pid from the kernel at all. The numbers are  
> >> used on the kernel-user boundary or within the kernel but  
> >> with the namespace this nr belongs to. This results in  
> >> massive changes of struct's members fro int pid to struct  
> >> pid \*pid, task->pid becomes the virtual id and so on and  
> >> so forth.  
> >> My approach is to keep the good old logic in the kernel.  
> >> The task->pid is a global and unique pid, find\_pid() finds  
> >> the pid by its global id and so on. The virtual ids appear  
> >> on the user-kernel boundary only. Thus drivers and other  
> >> kernel code may still be unaware of pids unless they do not  
> >> communicate with the userspace and get/put numerical pids.  
> >  
> > interesting ... not sure that is what kernel folks  
> > have in mind though (IIRC, the struct pid change was  
> > considered a kernel side cleanup)  
>  
> That's why I'm sending the patches - to make "kernel folks" make

> a decision. Will we see some patches from VServer team?

unlikely, as we do not require any pid virtualization except for the init pid (and blend through init)

but I'm worried about the fact that pid spaces will show up in the host context, which is usually not what the administrator likes to see ...  
(besides the fact that there probably is no way to tell what processes are real host processes at first glance, at least not with proper updates to ps and friends, which might be an option)

> >> And some more minor differences:

> >>

> >> 2. Suka's patches have the limit of pid namespace nesting.

> >> My patches do not.

> >>

> >> 3. Suka assumes that pid namespace can live without proc mount and tries to make the code work with pid\_ns->proc\_mnt change from NULL to not-NULL from times to times.

> >> My code calls the kern\_mount() at the namespace creation and thus the pid\_namespace always works with proc.

> >

> > shouldn't that be done by userspace instead?

>

> It can be. But when the namespace is being created there's no > any userspace in it yet.

I'm not talking about the 'userspace inside the space'  
I'm talking about the userspace creating the space  
(what if I do not want to have any proc mount?)

> >> There are some small issues that I can describe if someone is > >> interested.

> >>

> >> The tests like nptl perf, unixbench spawn, getpid and others > >> didn't reveal any performance degradation in init\_namespace > >> with the RHEL5 kernel .config file. I admit, that different > >> .config-s may show that patches hurt the performance, but the > >> intention was \*not\* to make the kernel work worse with popular > >> distributions.

> >>

> >> This set has some ways to move forward, but this is some kind > >> of a core, that do not change the init\_pid\_namespace behavior > >> (checked with LTP tests) and may require some hacking to do > >> with the namespaces only.

> >

> > TIA,  
> > Herbert  
>  
> BTW, why did you remove Suka and Serge from Cc?

once again, I do NOT remove anybody unless explicitly asked to do so, but I can do nothing against a broken mailing list ...

(so please go figure where the CC got lost, if you are sure you added it in the first place)

here the headers:

>From containers-bounces@lists.linux-foundation.org Fri Jul 6 10:03:04 2007  
Return-Path: containers-bounces@lists.linux-foundation.org  
X-Original-To: herbert@13thfloor.at  
Delivered-To: herbert@13thfloor.at  
Received: from smtp2.linux-foundation.org (smtp2.linux-foundation.org  
+[207.189.120.14])  
    (using TLSv1 with cipher DHE-RSA-AES256-SHA (256/256 bits))  
    (No client certificate requested)  
    by mail.13thfloor.at (Postfix) with ESMTP id 18186702C9  
    for <herbert@13thfloor.at>; Fri, 6 Jul 2007 10:02:35 +0200 (CEST)  
Received: from murdock.linux-foundation.org (localhost [127.0.0.1])  
    by smtp2.linux-foundation.org (8.13.5.20060308/8.13.5/Debian-3ubuntu1.1)+with ESMTP id  
I6682UJJ009593;  
    Fri, 6 Jul 2007 01:02:32 -0700  
Received: from relay.sw.ru (mailhub.sw.ru [195.214.233.200])  
    by smtp2.linux-foundation.org  
    (8.13.5.20060308/8.13.5/Debian-3ubuntu1.1) with ESMTP id  
I6682PXL009585  
    (version=TLSv1/SSLv3 cipher=DHE-RSA-AES256-SHA bits=256 verify=NO)  
    for <containers@lists.osdl.org>; Fri, 6 Jul 2007 01:02:28 -0700  
Received: from [192.168.3.76] ([192.168.3.76])  
    by relay.sw.ru (8.13.4/8.13.4) with ESMTP id I6681xfW003026;  
    Fri, 6 Jul 2007 12:02:00 +0400 (MSD)  
Message-ID: <468DF6F7.1010906@openvz.org>  
Date: Fri, 06 Jul 2007 12:01:59 +0400  
From: Pavel Emelianov <xemul@openvz.org>  
User-Agent: Thunderbird 1.5 (X11/20060317)  
MIME-Version: 1.0  
To: Andrew Morton <akpm@osdl.org>  
Content-Type: text/plain; charset=ISO-8859-1  
Content-Transfer-Encoding: 7bit  
Received-SPF: pass (localhost is always allowed.)  
X-Spam-Status: No, hits=-3.923 required=5  
+tests=AWL,BAYES\_00,OSDL\_HEADER\_SUBJECT\_BRACKETED

X-Spam-Checker-Version: SpamAssassin 3.1.0-osdl\_revision\_\_1.12\_\_  
X-MIMEDefang-Filter: osdl\$Revision: 1.181 \$  
X-Scanned-By: MIMEDefang 2.53 on 207.189.120.22  
Cc: Kirill Korotaev <dev@openvz.org>,  
Linux Kernel Mailing List <linux-kernel@vger.kernel.org>,  
"Eric W. Biederman" <ebiederm@xmission.com>,  
Linux Containers <containers@lists.osdl.org>  
Subject: [PATCH 0/16] Pid namespaces  
X-BeenThere: containers@lists.linux-foundation.org  
X-Mailman-Version: 2.1.5  
Precedence: list  
List-Id: Linux Containers <containers.lists.linux-foundation.org>  
List-Unsubscribe:  
+<https://lists.linux-foundation.org/mailman/listinfo/containers>,  
+<mailto:containers-request@lists.linux-foundation.org?subject=unsubscribe>  
List-Archive: <http://lists.linux-foundation.org/pipermail/containers>  
List-Post: <mailto:containers@lists.linux-foundation.org>  
List-Help: <mailto:containers-request@lists.linux-foundation.org?subject=help>  
List-Subscribe:  
+<https://lists.linux-foundation.org/mailman/listinfo/containers>,  
<mailto:containers-request@lists.linux-foundation.org?subject=subscribe>Sender:  
containers-bounces@lists.linux-foundation.org  
Errors-To: containers-bounces@lists.linux-foundation.org  
Status: RO  
X-Status: A  
Content-Length: 2788  
Lines: 64

>  
> Pavel

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Cedric Le Goater](#) on Mon, 09 Jul 2007 20:06:24 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Badari Pulavarty wrote:  
> On Fri, 2007-07-06 at 12:01 +0400, Pavel Emelianov wrote:  
>> This is "submission for inclusion" of hierarchical, not kconfig  
>> configurable, zero overheaded ;) pid namespaces.  
>  
> Not able to boot my ppc64 machine with the patchset :(

I can't boot either on a x86\_64 but I don't even have logs to send :(

C.

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Cedric Le Goater](#) on Mon, 09 Jul 2007 20:12:01 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

>>>> 3. Suka assumes that pid namespace can live without proc mount  
>>>> and tries to make the code work with pid\_ns->proc\_mnt change  
>>>> from NULL to not-NULL from times to times.  
>>>> My code calls the kern\_mount() at the namespace creation and  
>>>> thus the pid\_namespace always works with proc.  
>>> shouldn't that be done by userspace instead?  
>> It can be. But when the namespace is being created there's no  
>> any userspace in it yet.  
>  
> I'm not talking about the 'userspace inside the space'  
> I'm talking about the userspace creating the space  
> (what if I do not want to have any proc mount?)

yes, can't we let the user doing the unshare or clone decide whether  
it needs to mount /proc or not in the new pid namespace ?

that's already optional on the host.

C.

---

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

---

---

Subject: Re: [PATCH 1/16] Round up the API  
Posted by [Cedric Le Goater](#) on Mon, 09 Jul 2007 20:18:33 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Pavel Emelianov wrote:

> The set of functions process\_session, task\_session, process\_group  
> and task\_pgrp is confusing, as the names can be mixed with each other  
> when looking at the code for a long time.

>  
> The proposals are to  
> \* equip the functions that return the integer with \_nr suffix to  
> represent that fact,  
> \* and to make all functions work with task (not process) by making  
> the common prefix of the same name.  
>  
> For monotony the routines signal\_session() and set\_signal\_session()  
> are replaced with task\_session\_nr() and set\_task\_session(), especially  
> since they are only used with the explicit task->signal dereference.  
>  
> Signed-off-by: Pavel Emelianov <xemul@openvz.org>  
> Acked-by: Serge E. Hallyn <serue@us.ibm.com>

please let's get that one in.

I think we are all ok with it. right ?

C.

---

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

---

---

Subject: Re: [PATCH 2/16] Miscellaneous preparations for namespaces  
Posted by [Cedric Le Goater](#) on Mon, 09 Jul 2007 20:22:13 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Pavel Emelianov wrote:  
> The most important change is moving exit\_task\_namespaces()  
> inside exit\_notify() to makes it possible to notify the  
> exiting task's parent. However this should be done before  
> release\_task() to address the issue pointed by Sukadev with  
> NFS kernel thread.

Have you actually checked that doing an unshare() with a NFS mount ?

> Other changes are small and do not deserve separate description.

yes. if they were in a separate patch, you could push them to -mm.

thanks,

C.

---

Containers mailing list  
Containers@lists.linux-foundation.org

---



---

Subject: Re: [PATCH 4/16] Change data structures for pid namespaces

Posted by [Cedric Le Goater](#) on Mon, 09 Jul 2007 20:25:41 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Pavel Emelianov wrote:

```
> struct pid_namespace will have the kmem_cache to allocate
> the pids from, the parent, as they are hierarchical, and
> the level of nesting value.
>
> struct pid will have a variable length array of pid_number-s
> one for each namespace this pid lives in. The level value
> shows the level of the namespace this pid lives in and thus -
> the number of elements in the numbers array.
>
> Signed-off-by: Pavel Emelianov <xemul@openvz.org>
>
> ---
>
> include/linux/init_task.h | 6 ++++++
> include/linux/pid.h       | 9 ++++++++
> include/linux/pid_namespace.h | 3 +++
> kernel/pid.c             | 3 +-
> 4 files changed, 20 insertions(+), 1 deletion(-)
>
> diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid.h linux-2.6.22-rc4-mm2-2/include/linux/pid.h
> --- linux-2.6.22-rc4-mm2.orig/include/linux/pid.h 2007-06-14 12:14:29.000000000 +0400
> +++ linux-2.6.22-rc4-mm2-2/include/linux/pid.h 2007-07-04 19:00:38.000000000 +0400
> @@ -40,6 +40,13 @@ enum pid_type
>  * processes.
>  */
>
> +struct pid_number {
> + /* Try to keep pid_chain in the same cacheline as nr for find_pid */
> + int nr;
> + struct pid_namespace *ns;
> + struct hlist_node pid_chain;
> +};
> +
> struct pid
> {
> atomic_t count;
> @@ -40,6 +40,8 @@ enum pid_type
>  /* lists of tasks that use this pid */
> struct hlist_head tasks[PIDTYPE_MAX];
> struct rcu_head rcu;
```

```

> + int level;
> + struct pid_number numbers[1];
> };
>
> extern struct pid init_struct_pid;
> diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid_namespace.h
linux-2.6.22-rc4-mm2-2/include/linux/pid_namespace.h
> --- linux-2.6.22-rc4-mm2.orig/include/linux/pid_namespace.h 2007-06-14 12:14:29.000000000
+0400
> +++ linux-2.6.22-rc4-mm2-2/include/linux/pid_namespace.h 2007-07-04 19:00:39.000000000
+0400
> @@ -16,7 +15,10 @@ struct pidmap {
> struct kref kref;
> struct pidmap pidmap[PIDMAP_ENTRIES];
> int last_pid;
> + int level;
> struct task_struct *child_reaper;
> + struct kmem_cache *pid_cachep;

```

so, that looks like a good idea to have the cache in the pidmap. could you push that independently to see how it all fits together ?

thanks,

C.

---

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

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Sukadev Bhattiprolu](#) on Mon, 09 Jul 2007 21:42:44 GMT  
[View Forum Message](#) <> [Reply to Message](#)

Pavel Emelianov [xemul@openvz.org] wrote:

```

| This is "submission for inclusion" of hierarchical, not kconfig
| configurable, zero overheaded ;) pid namespaces.
|
| The overall idea is the following:
|
| The namespace are organized as a tree - once a task is cloned
| with CLONE_NEWPIDS (yes, I've also switched to it :) the new
| namespace becomes the parent's child and tasks living in the
| parent namespace see the tasks from the new one. The numerical
| ids are used on the kernel-user boundary, i.e. when we export
| pid to user we show the id, that should be used to address the

```

| task in question from the namespace we're exporting this id to.

| The main difference from Suka's patches are the following:

- | 0. Suka's patches change the kernel/pid.c code too heavy.  
| This set keeps the kernel code look like it was without  
| the patches. However, this is a minor issue. The major is:
- | 1. Suka's approach is to remove the notion of the task's  
| numerical pid from the kernel at all. The numbers are  
| used on the kernel-user boundary or within the kernel but  
| with the namespace this nr belongs to. This results in  
| massive changes of struct's members fro int pid to struct  
| pid \*pid, task->pid becomes the virtual id and so on and  
| so forth.

Your basic design is similar to what our patchset has been for  
a while, with a few changes.

My patchset does not remove the task->pid. It still uses it  
with the caveat that with multiple namespaces it is not unique.  
getpid() implementation does not changes for instance.

Basically our patchset has init\_pid\_ns as the last element in the  
pid->numbers[] array while yours is having it as the first. How  
big a difference it makes, I am not sure.

| My approach is to keep the good old logic in the kernel.  
| The task->pid is a global and unique pid, find\_pid() finds  
| the pid by its global id and so on. The virtual ids appear  
| on the user-kernel boundary only. Thus drivers and other  
| kernel code may still be unaware of pids unless they do not  
| communicate with the userspace and get/put numerical pids.

Even in my patchset, drivers or other kernel code have no need  
to know anything about namespaces.

Actually you seem to introduce a new function find\_vpid() that  
is used in a driver. So a driver-writer needs to know whether  
to call find\_pid() or find\_vpid().

| And some more minor differences:

- | 2. Suka's patches have the limit of pid namespace nesting.  
| My patches do not.

Yes - its a compile-time constant (MAX\_NESTED\_PID\_NS) that I introduced just in the last version to simplify allocation. Especially after you argued against arbitrary depth before :-)

The basic design of your new 'struct pid' data structure is very similar to what we have had for the last couple of rounds and we could just as easily remove MAX\_NESTED\_PID\_NS.

|  
| 3. Suka assumes that pid namespace can live without proc mount  
| and tries to make the code work with pid\_ns->proc\_mnt change  
| from NULL to not-NULL from times to times.  
| My code calls the kern\_mount() at the namespace creation and  
| thus the pid\_namespace always works with proc.

Yes, we have been debating about the better approach for this yet. We have been considering doing the kern\_mount, as we do in init\_pid\_ns at present.

|  
| There are some small issues that I can describe if someone is  
| interested.

| The tests like nptl perf, unixbench spawn, getpid and others  
| didn't reveal any performance degradation in init\_namespace  
| with the RHEL5 kernel .config file. I admit, that different  
| .config-s may show that patches hurt the performance, but the  
| intention was \*not\* to make the kernel work worse with popular  
| distributions.

| This set has some ways to move forward, but this is some kind  
| of a core, that do not change the init\_pid\_namespace behavior  
| (checked with LTP tests) and may require some hacking to do  
| with the namespaces only.

| Patches apply to 2.6.22-rc6-mm1.

---

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

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Badari Pulavarty](#) on Mon, 09 Jul 2007 23:00:34 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Mon, 2007-07-09 at 22:06 +0200, Cedric Le Goater wrote:  
> Badari Pulavarty wrote:

> > On Fri, 2007-07-06 at 12:01 +0400, Pavel Emelianov wrote:  
> >> This is "submission for inclusion" of hierarchical, not kconfig  
> >> configurable, zero overheaded ;) pid namespaces.  
> >  
> > Not able to boot my ppc64 machine with the patchset :(  
>  
> I can't boot either on a x86\_64 but I don't even have logs to send :(

Yes. It blew up way early in the boot on my x86\_64, so nothing came up on the console to capture (blank screen) :(

Thanks,  
Badari

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Sukadev Bhattiprolu](#) on Tue, 10 Jul 2007 00:29:29 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Pavel Emelianov [xemul@openvz.org] wrote:  
| This is "submission for inclusion" of hierarchical, not kconfig  
| configurable, zero overheaded ;) pid namespaces.  
|  
| The overall idea is the following:  
|  
| The namespace are organized as a tree - once a task is cloned  
| with CLONE\_NEWPIDS (yes, I've also switched to it :) the new

Can you really clone() a pid namespace all by itself ?  
copy\_namespaces() has the following:

```
    if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |  
CLONE_NEWUSER)))  
        return 0;
```

doesn't it mean you cannot create a pid namespace using clone() unless one of the above flags are also specified ?

unshare\_nsproxy\_namespaces() has the following correct check:

```
    if (!(unshare_flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |  
CLONE_NEWUSER | CLONE_NEWPIDS)))
```

```
return 0;
```

BTW, why not use CLONE\_NEWPID and drop the 'S' ? We don't have 'S' with other namespaces.

Suka

---

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

---

---

Subject: Re: [PATCH 7/16] Helpers to find the task by its numerical ids  
Posted by [Sukadev Bhattiprolu](#) on Tue, 10 Jul 2007 04:00:45 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Pavel Emelianov [xemul@openvz.org] wrote:

```
| When searching the task by numerical id one may need to find  
| it using global pid (as it is done now in kernel) or by its  
| virtual id, e.g. when sending a signal to a task from one  
| namespace the sender will specify the task's virtual id.
```

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

```
| ---
```

```
| fs/proc/base.c      | 2 +-  
| include/linux/pid.h | 13 ++++++++  
| include/linux/sched.h | 31 ++++++++  
| kernel/pid.c        | 32 ++++++++  
| 4 files changed, 58 insertions(+), 20 deletions(-)
```

```
| --- ./fs/proc/base.c.ve6 2007-07-06 10:58:56.000000000 +0400
```

```
| +++ ./fs/proc/base.c 2007-07-06 11:03:41.000000000 +0400
```

```
| @@ -2230,7 +2230,7 @@ static struct task_struct *next_tgid(uns  
| rcu_read_lock());
```

```
| retry:
```

```
| task = NULL;
```

```
| - pid = find_ge_pid(tgid);
```

```
| + pid = find_ge_pid(tgid, &init_pid_ns);
```

```
| if (pid) {
```

```
| tgid = pid->nr + 1;
```

```
| task = pid_task(pid, PIDTYPE_PID);
```

```
| --- ./include/linux/pid.h.ve6 2007-07-06 11:03:27.000000000 +0400
```

```
| +++ ./include/linux/pid.h 2007-07-06 11:03:27.000000000 +0400
```

```
| @@ -98,14 +98,23 @@ extern struct pid_namespace init_pid_ns;
```

```
| /*
```

```
| * look up a PID in the hash table. Must be called with the tasklist_lock
```

```

| * or rcu_read_lock() held.
| + *
| + * find_pid_ns() finds the pid in the namespace specified
| + * find_pid() find the pid by its global id, i.e. in the init namespace
| + * find_vpid() find the pid by its virtual id, i.e. in the current namespace
| + *
| + * see also find_task_by_pid() set in include/linux/sched.h
| */
| -extern struct pid *FASTCALL(find_pid(int nr));
| +extern struct pid *FASTCALL(find_pid_ns(int nr, struct pid_namespace *ns));
| +
| + #define find_vpid(pid) find_pid_ns(pid, current->nsproxy->pid_ns)
| + #define find_pid(pid) find_pid_ns(pid, &init_pid_ns)

```

Adding a second interface maybe more confusing to drivers and non-pid users.

But more importantly, modifying find\_pid() to refer to only init\_pid\_ns would require auditing existing find\_pid() callers and switching them to find\_vpid().

For instance if capset() is called from a child pid namespace, the 'pid' would refer to the pid or pgid from child pid ns. But cap\_set\_pg() calls find\_pid() which gets the number from init\_pid\_ns.

Is there a similar issue with sunos\_killpg() ?

---

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

---

Subject: Re: [PATCH 8/16] Masquerade the siginfo when sending a pid to a foreign namespace

Posted by [Sukadev Bhattiprolu](#) on Tue, 10 Jul 2007 04:18:00 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Pavel Emelianov [xemul@openvz.org] wrote:

```

| When user send signal from (say) init namespace to any task in a sub
| namespace the siginfo struct must not carry the sender's pid value, as
| this value may refer to some task in the destination namespace and thus
| may confuse the application.

```

Also, do you prevent signals to the child reaper of a container from within its container ? If so, can you show me where you handle it ? I can't seem to find it.

And I guess you do allow signals to the child-reaper of a container from

its parent container.

| The consensus was to pretend in this case as if it is the kernel who  
| sends the signal.

| The pid\_ns\_accessible() call is introduced to check this pid-to-ns  
| accessibility.

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

| ---

| include/linux/pid.h | 10 ++++++++  
| kernel/signal.c | 34 ++++++-----  
| 2 files changed, 38 insertions(+), 6 deletions(-)

| diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid.h linux-2.6.22-rc4-mm2-2/include/linux/pid.h  
| --- linux-2.6.22-rc4-mm2.orig/include/linux/pid.h 2007-06-14 12:14:29.000000000 +0400  
| +++ linux-2.6.22-rc4-mm2-2/include/linux/pid.h 2007-07-04 19:00:38.000000000 +0400  
| @@ -83,6 +89,16 @@ extern void FASTCALL(detach\_pid(struct t  
| return nr;  
| }  
|

| +/\*  
| + \* checks whether the pid actually lives in the namespace ns, i.e. it was  
| + \* created in this namespace or it was moved there.  
| + \*/  
| +  
| +static inline int pid\_ns\_accessible(struct pid\_namespace \*ns, struct pid \*pid)  
| +{  
| + return pid->numbers[pid->level].ns == ns;  
| +}  
| +

| #define do\_each\_pid\_task(pid, type, task) \  
| do { \  
| struct hlist\_node \*pos\_\_\_; \  
| diff -upr linux-2.6.22-rc4-mm2.orig/kernel/signal.c linux-2.6.22-rc4-mm2-2/kernel/signal.c  
| --- linux-2.6.22-rc4-mm2.orig/kernel/signal.c 2007-07-04 19:00:38.000000000 +0400  
| +++ linux-2.6.22-rc4-mm2-2/kernel/signal.c 2007-07-04 19:00:38.000000000 +0400  
| @@ -1124,13 +1124,31 @@ EXPORT\_SYMBOL\_GPL(kill\_pid\_info\_as\_uid);  
| \* is probably wrong. Should make it like BSD or SYSV.  
| \*/  
|

| -static int kill\_something\_info(int sig, struct siginfo \*info, int pid)  
| +static inline void masquerade\_siginfo(struct pid\_namespace \*src\_ns,  
| + struct pid \*tgt\_pid, struct siginfo \*info)  
| +{



```

| + if (tgt_pid != NULL && !pid_ns_accessible(src_ns, tgt_pid)) {
| + /*
| +  * current namespace is not seen from the taks we
| +  * want to send the signal to, so pretend as if it
| +  * is the kernel who does this to avoid pid messing
| +  * by the target
| +  */
| +
| + info->si_pid = 0;
| + info->si_code = SI_KERNEL;
| + }
| +}
| +
| +static int kill_something_info(int sig, struct siginfo *info, int pid_nr)
| {
|   int ret;
| + struct pid *pid;
| +
|   rcu_read_lock();
| - if (!pid) {
| + if (!pid_nr) {
|     ret = kill_pgrp_info(sig, info, task_pgrp(current));
| - } else if (pid == -1) {
| + } else if (pid_nr == -1) {
|     int retval = 0, count = 0;
|     struct task_struct * p;

```

So what happens if we run "kill -s <sig> -1" from within a container ?  
Do you terminate all processes in the system or just the process in  
the container ?

---

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

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Sukadev Bhattiprolu](#) on Tue, 10 Jul 2007 04:26:19 GMT  
[View Forum Message](#) <> [Reply to Message](#)

I am not able to find a specific patch that this might be in,  
but what happens when the child-reaper of a container exits ?  
Do you terminate all processes in the container ? I thought  
that was discussed earlier and the consensus was to terminate  
all processes in that container and its subordinate containers.

Is that not the case now ?

## Suka

Pavel Emelianov [xemul@openvz.org] wrote:

| This is "submission for inclusion" of hierarchical, not kconfig  
| configurable, zero overheaded ;) pid namespaces.

| The overall idea is the following:

| The namespace are organized as a tree - once a task is cloned  
| with CLONE\_NEWPIDS (yes, I've also switched to it :) the new  
| namespace becomes the parent's child and tasks living in the  
| parent namespace see the tasks from the new one. The numerical  
| ids are used on the kernel-user boundary, i.e. when we export  
| pid to user we show the id, that should be used to address the  
| task in question from the namespace we're exporting this id to.

| The main difference from Suka's patches are the following:

- | 0. Suka's patches change the kernel/pid.c code too heavy.  
| This set keeps the kernel code look like it was without  
| the patches. However, this is a minor issue. The major is:
- | 1. Suka's approach is to remove the notion of the task's  
| numerical pid from the kernel at all. The numbers are  
| used on the kernel-user boundary or within the kernel but  
| with the namespace this nr belongs to. This results in  
| massive changes of struct's members fro int pid to struct  
| pid \*pid, task->pid becomes the virtual id and so on and  
| so forth.  
| My approach is to keep the good old logic in the kernel.  
| The task->pid is a global and unique pid, find\_pid() finds  
| the pid by its global id and so on. The virtual ids appear  
| on the user-kernel boundary only. Thus drivers and other  
| kernel code may still be unaware of pids unless they do not  
| communicate with the userspace and get/put numerical pids.

| And some more minor differences:

- | 2. Suka's patches have the limit of pid namespace nesting.  
| My patches do not.
- | 3. Suka assumes that pid namespace can live without proc mount  
| and tries to make the code work with pid\_ns->proc\_mnt change  
| from NULL to not-NULL from times to times.  
| My code calls the kern\_mount() at the namespace creation and  
| thus the pid\_namespace always works with proc.

| There are some small issues that I can describe if someone is

| interested.

| The tests like nptl perf, unixbench spawn, getpid and others  
| didn't reveal any performance degradation in init\_namespace  
| with the RHEL5 kernel .config file. I admit, that different  
| .config-s may show that patches hurt the performance, but the  
| intention was \*not\* to make the kernel work worse with popular  
| distributions.

| This set has some ways to move forward, but this is some kind  
| of a core, that do not change the init\_pid\_namespace behavior  
| (checked with LTP tests) and may require some hacking to do  
| with the namespaces only.

| Patches apply to 2.6.22-rc6-mm1.

---

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

---

---

Subject: Re: [PATCH 4/16] Change data structures for pid namespaces

Posted by [Sukadev Bhattiprolu](#) on Tue, 10 Jul 2007 04:32:50 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Cedric Le Goater [clg@fr.ibm.com] wrote:

| Pavel Emelianov wrote:

| > struct pid\_namespace will have the kmem\_cache to allocate  
| > the pids from, the parent, as they are hierarchical, and  
| > the level of nesting value.

| >

| > struct pid will have a variable length array of pid\_number-s  
| > one for each namespace this pid lives in. The level value  
| > shows the level of the namespace this pid lives in and thus -  
| > the number of elements in the numbers array.

| >

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

| >

| > ---

| >

| > include/linux/init\_task.h | 6 ++++++

| > include/linux/pid.h | 9 ++++++++

| > include/linux/pid\_namespace.h | 3 +++

| > kernel/pid.c | 3 ++-

| > 4 files changed, 20 insertions(+), 1 deletion(-)

| >

| > diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid.h

linux-2.6.22-rc4-mm2-2/include/linux/pid.h

```

| > --- linux-2.6.22-rc4-mm2.orig/include/linux/pid.h 2007-06-14 12:14:29.000000000 +0400
| > +++ linux-2.6.22-rc4-mm2-2/include/linux/pid.h 2007-07-04 19:00:38.000000000 +0400
| > @@ -40,6 +40,13 @@ enum pid_type
| > * processes.
| > */
| >
| > +struct pid_number {
| > + /* Try to keep pid_chain in the same cacheline as nr for find_pid */
| > + int nr;
| > + struct pid_namespace *ns;
| > + struct hlist_node pid_chain;
| > +};

```

We meant to go back and look at removing the extra 'struct pid \*' we had here. Looks like you did that. Cool.

```

| > +
| > struct pid
| > {
| > atomic_t count;
| > @@ -40,6 +40,8 @@ enum pid_type
| > /* lists of tasks that use this pid */
| > struct hlist_head tasks[PIDTYPE_MAX];
| > struct rcu_head rcu;
| > + int level;
| > + struct pid_number numbers[1];
| > };
| >
| > extern struct pid init_struct_pid;
| > diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid_namespace.h
linux-2.6.22-rc4-mm2-2/include/linux/pid_namespace.h
| > --- linux-2.6.22-rc4-mm2.orig/include/linux/pid_namespace.h 2007-06-14 12:14:29.000000000
+0400
| > +++ linux-2.6.22-rc4-mm2-2/include/linux/pid_namespace.h 2007-07-04 19:00:39.000000000
+0400
| > @@ -16,7 +15,10 @@ struct pidmap {
| > struct kref kref;
| > struct pidmap pidmap[PIDMAP_ENTRIES];
| > int last_pid;
| > + int level;
| > struct task_struct *child_reaper;
| > + struct kmem_cache *pid_cachep;
|
| so, that looks like a good idea to have the cache in the pidmap. could you
| push that independently to see how it all fits together ?

```

Yes. I like this idea too.

|  
| thanks,  
|  
| C.

---

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

---

---

Subject: Re: [PATCH 6/16] Helpers to obtain pid numbers  
Posted by [Sukadev Bhattiprolu](#) on Tue, 10 Jul 2007 05:18:01 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Pavel Emelianov [xemul@openvz.org] wrote:  
| When showing pid to user or getting the pid numerical id for in-kernel  
| use the value of this id may differ depending on the namespace.

| This set of helpers is used to get the global pid nr, the virtual (i.e.  
| seen by task in its namespace) nr and the nr as it is seen from the  
| specified namespace.

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

| ---

| include/linux/pid.h | 27 ++++++++  
| include/linux/sched.h | 108 ++++++++++++++++++++++++++++++++++++++-----  
| kernel/pid.c | 8 +++  
| 3 files changed, 132 insertions(+), 11 deletions(-)

| diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid.h linux-2.6.22-rc4-mm2-2/include/linux/pid.h  
| --- linux-2.6.22-rc4-mm2.orig/include/linux/pid.h 2007-06-14 12:14:29.000000000 +0400  
| +++ linux-2.6.22-rc4-mm2-2/include/linux/pid.h 2007-07-04 19:00:38.000000000 +0400  
| @@ -83,6 +89,9 @@ extern void FASTCALL(detach\_pid(struct t  
| extern void FASTCALL(transfer\_pid(struct task\_struct \*old,  
| struct task\_struct \*new, enum pid\_type));  
| +struct pid\_namespace;  
| +extern struct pid\_namespace init\_pid\_ns;  
| +  
| /\*  
| \* look up a PID in the hash table. Must be called with the tasklist\_lock  
| \* or rcu\_read\_lock() held.  
| @@ -93,14 +99,36 @@ extern void FASTCALL(detach\_pid(struct t  
| extern struct pid \*alloc\_pid(void);  
| extern void FASTCALL(free\_pid(struct pid \*pid));

```

| +/*
| + * the helpers to get the pid's id seen from different namespaces
| + *
| + * pid_nr() : global id, i.e. the id seen from the init namespace;
| + * pid_vnr() : virtual id, i.e. the id seen from the namespace this pid
| + *           belongs to. this only makes sense when called in the
| + *           context of the task that belongs to the same namespace;
| + * pid_nr_ns() : id seen from the ns specified.
| + *
| + * see also task_xid_nr() etc in include/linux/sched.h
| + */

```

I think its a bit confusing and error-prone to have both pid\_nr() and pid\_vnr().

BTW, shouldn't you use pid\_vnr() in do\_task\_stat() ? You currently use pid\_nr() and that returns the init-pid-ns id right ?

```

| +
| static inline pid_t pid_nr(struct pid *pid)
| {
| pid_t nr = 0;
| if (pid)
| - nr = pid->nr;
| + nr = pid->numbers[0].nr;
| return nr;
| }
|
| +pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns);
| +
| +static inline pid_t pid_vnr(struct pid *pid)
| +{
| + pid_t nr = 0;
| + if (pid)
| + nr = pid->numbers[pid->level].nr;
| + return nr;
| +}
| +
| #define do_each_pid_task(pid, type, task) \
| do { \
| struct hlist_node *pos___; \
diff -upr linux-2.6.22-rc4-mm2.orig/kernel/pid.c linux-2.6.22-rc4-mm2-2/kernel/pid.c
--- linux-2.6.22-rc4-mm2.orig/kernel/pid.c 2007-06-14 12:14:29.000000000 +0400
+++ linux-2.6.22-rc4-mm2-2/kernel/pid.c 2007-07-04 19:00:38.000000000 +0400
| @@ -339,6 +379,14 @@ struct pid *find_get_pid(pid_t nr)
| return pid;
| }
|

```

```

| +pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
| +{
| + pid_t nr = 0;
| + if (pid && ns->level <= pid->level)
| + nr = pid->numbers[ns->level].nr;
| + return nr;
| +}
| +
| /*
|  * Used by proc to find the first pid that is greater then or equal to nr.
|  *
| diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/sched.h
linux-2.6.22-rc4-mm2-2/include/linux/sched.h
| --- linux-2.6.22-rc4-mm2.orig/include/linux/sched.h 2007-07-04 19:00:38.000000000 +0400
| +++ linux-2.6.22-rc4-mm2-2/include/linux/sched.h 2007-07-04 19:00:38.000000000 +0400
| @@ -1153,16 +1154,6 @@ struct task_struct {
| #endif
| };
|
| -static inline pid_t task_pgrp_nr(struct task_struct *tsk)
| -{
| - return tsk->signal->pgrp;
| -}
| -
| -static inline pid_t task_session_nr(struct task_struct *tsk)
| -{
| - return tsk->signal->__session;
| -}
| -
| static inline void set_task_session(struct task_struct *tsk, pid_t session)
| {
| tsk->signal->__session = session;
| @@ -1188,6 +1179,104 @@ static inline struct pid *task_session(s
| return task->group_leader->pids[PIDTYPE_SID].pid;
| }
|
| +struct pid_namespace;
| +
| +/*
| + * the helpers to get the task's different pids as they are seen
| + * from various namespaces
| + *
| + * task_xid_nr()    : global id, i.e. the id seen from the init namespace;
| + * task_xid_vnr()  : virtual id, i.e. the id seen from the namespace the task
| + *                  belongs to. this only makes sence when called in the
| + *                  context of the task that belongs to the same namespace;
| + * task_xid_nr_ns() : id seen from the ns specified;
| + *

```

```

| + * set_task_vxid() : assigns a virtual id to a task;
| + *
| + * task_ppid_nr_ns() : the parent's id as seen from the namespace specified.
| + *           the result depends on the namespace and whether the
| + *           task in question is the namespace's init. e.g. for the
| + *           namespace's init this will return 0 when called from
| + *           the namespace of this init, or appropriate id otherwise.
| + *
| + *
| + * see also pid_nr() etc in include/linux/pid.h
| + */
| +
| +static inline pid_t task_pid_nr(struct task_struct *tsk)
| +{
| + return tsk->pid;
| +}
| +
| +static inline pid_t task_pid_nr_ns(struct task_struct *tsk,
| + struct pid_namespace *ns)
| +{
| + return pid_nr_ns(task_pid(tsk), ns);
| +}
| +
| +static inline pid_t task_pid_vnr(struct task_struct *tsk)
| +{
| + return pid_vnr(task_pid(tsk));
| +}
| +
| +
| +static inline pid_t task_tgid_nr(struct task_struct *tsk)
| +{
| + return tsk->tgid;
| +}
| +
| +static inline pid_t task_tgid_nr_ns(struct task_struct *tsk,
| + struct pid_namespace *ns)
| +{
| + return pid_nr_ns(task_tgid(tsk), ns);
| +}
| +
| +static inline pid_t task_tgid_vnr(struct task_struct *tsk)
| +{
| + return pid_vnr(task_tgid(tsk));
| +}
| +
| +
| +static inline pid_t task_pgrp_nr(struct task_struct *tsk)
| +{

```



```

| + return tsk->signal->pgrp;
| +}
| +
| +static inline pid_t task_pgrp_nr_ns(struct task_struct *tsk,
| + struct pid_namespace *ns)
| +{
| + return pid_nr_ns(task_pgrp(tsk), ns);
| +}
| +
| +
| +static inline pid_t task_pgrp_vnr(struct task_struct *tsk)
| +{
| + return pid_vnr(task_pgrp(tsk));
| +}
| +
| +
| +
| +static inline pid_t task_session_nr(struct task_struct *tsk)
| +{
| + return tsk->signal->__session;
| +}
| +
| +
| +static inline pid_t task_session_nr_ns(struct task_struct *tsk,
| + struct pid_namespace *ns)
| +{
| + return pid_nr_ns(task_session(tsk), ns);
| +}
| +
| +
| +
| +static inline pid_t task_session_vnr(struct task_struct *tsk)
| +{
| + return pid_vnr(task_session(tsk));
| +}
| +
| +
| +
| +static inline pid_t task_ppid_nr_ns(struct task_struct *tsk,
| + struct pid_namespace *ns)
| +{
| + return pid_nr_ns(task_pid(rcu_dereference(tsk->real_parent)), ns);
| +}
| +
| +
| +/**
| + * pid_alive - check that a task structure is not stale
| + * @p: Task structure to be checked.

```

---

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

---

Subject: Re: [PATCH 1/16] Round up the API  
Posted by [Pavel Emelianov](#) on Tue, 10 Jul 2007 06:40:13 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Cedric Le Goater wrote:

> Pavel Emelianov wrote:

>> The set of functions process\_session, task\_session, process\_group  
>> and task\_pgrp is confusing, as the names can be mixed with each other  
>> when looking at the code for a long time.

>>

>> The proposals are to

>> \* equip the functions that return the integer with \_nr suffix to

>> represent that fact,

>> \* and to make all functions work with task (not process) by making

>> the common prefix of the same name.

>>

>> For monotony the routines signal\_session() and set\_signal\_session()

>> are replaced with task\_session\_nr() and set\_task\_session(), especially

>> since they are only used with the explicit task->signal dereference.

>>

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

>> Acked-by: Serge E. Hallyn <serue@us.ibm.com>

>

> please let's get that one in.

>

> I think we are all ok with it. right ?

Right. That's already the 3rd time I send it to Andrew...

> C.

>

---

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>

---

---

Subject: Re: [PATCH 2/16] Miscelaneous preparations for namespaces

Posted by [Pavel Emelianov](#) on Tue, 10 Jul 2007 06:42:09 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Cedric Le Goater wrote:

> Pavel Emelianov wrote:

>> The most important change is moving exit\_task\_namespaces()

>> inside exit\_notify() to makes it possible to notify the

>> exiting task's parent. However this should be done before

>> release\_task() to address the issue pointed by Sukadev with

>> NFS kernel thread.

>

> Have you actually checked that doing an unshare() with a NFS mount ?

Not unshare(), but clone(). I admit that I lost smth significant, but everything was fine...

>> Other changes are small and do not deserve separate description.

>

> yes. if they were in a separate patch, you could push them to -mm.

OK

> thanks,

>

> C.

>

---

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>

---

---

Subject: Re: [PATCH 7/16] Helpers to find the task by its numerical ids  
Posted by [Pavel Emelianov](#) on Tue, 10 Jul 2007 06:47:09 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

sukadev@us.ibm.com wrote:

> Pavel Emelianov [xemul@openvz.org] wrote:

> | When searching the task by numerical id one may need to find

> | it using global pid (as it is done now in kernel) or by its

> | virtual id, e.g. when sending a signal to a task from one

> | namespace the sender will specify the task's virtual id.

> |

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

> |

> | ---

> |

> | fs/proc/base.c | 2 +-  
> | include/linux/pid.h | 13 ++++++++  
> | include/linux/sched.h | 31 ++++++++  
> | kernel/pid.c | 32 ++++++++-----  
> | 4 files changed, 58 insertions(+), 20 deletions(-)

> |

> |

> |

> |

> | --- ./fs/proc/base.c.ve6 2007-07-06 10:58:56.000000000 +0400

> | +++ ./fs/proc/base.c 2007-07-06 11:03:41.000000000 +0400

> | @@ -2230,7 +2230,7 @@ static struct task\_struct \*next\_tgid(uns

```

> | rcu_read_lock();
> | retry:
> | task = NULL;
> | - pid = find_ge_pid(tgid);
> | + pid = find_ge_pid(tgid, &init_pid_ns);
> | if (pid) {
> |     tgid = pid->nr + 1;
> |     task = pid_task(pid, PIDTYPE_PID);
> | --- ./include/linux/pid.h.ve6 2007-07-06 11:03:27.000000000 +0400
> | +++ ./include/linux/pid.h 2007-07-06 11:03:27.000000000 +0400
> | @@ -98,14 +98,23 @@ extern struct pid_namespace init_pid_ns;
> | /*
> |  * look up a PID in the hash table. Must be called with the tasklist_lock
> |  * or rcu_read_lock() held.
> | + *
> | + * find_pid_ns() finds the pid in the namespace specified
> | + * find_pid() find the pid by its global id, i.e. in the init namespace
> | + * find_vpid() find the pid by its virtual id, i.e. in the current namespace
> | + *
> | + * see also find_task_by_pid() set in include/linux/sched.h
> | */
> | -extern struct pid *FASTCALL(find_pid(int nr));
> | +extern struct pid *FASTCALL(find_pid_ns(int nr, struct pid_namespace *ns));
> | +
> | + #define find_vpid(pid) find_pid_ns(pid, current->nsproxy->pid_ns)
> | + #define find_pid(pid) find_pid_ns(pid, &init_pid_ns)
>
> Adding a second interface maybe more confusing to drivers and non-pid
> users.
>
> But more importantly, modifying find_pid() to refer to only init_pid_ns
> would require auditing existing find_pid() callers and switching them to
> find_vpid().
>
> For instance if capset() is called from a child pid namespace, the 'pid'
> would refer to the pid or pgid from child pid ns. But cap_set_pg() calls
> find_pid() which gets the number from init_pid_ns.
>
> Is there a similar issue with sunos_killpg() ?
>

```

Yes, I know this. The [PATCH 15/16] has to switch all the kernel-to-user boundaries to use the additional helpers. That's the hardest part and I agree that I could lost something in it.

However, this is relevant only (!) when you clone the namespace. So people who do not need them won't suffer when this patch set is in mainline.

That's my intention - to make a set that doesn't affect the non-namespace-d case and go on polishing it. You have already pointed out 2 places. I expect people to find more of them. This is easier to patch only the boundary to the user rather than the whole kernel :)

Thanks,  
Pavel

---

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

---

---

Subject: Re: [PATCH 6/16] Helpers to obtain pid numbers  
Posted by [Pavel Emelianov](#) on Tue, 10 Jul 2007 06:49:58 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

sukadev@us.ibm.com wrote:  
> Pavel Emelianov [xemul@openvz.org] wrote:  
> | When showing pid to user or getting the pid numerical id for in-kernel  
> | use the value of this id may differ depending on the namespace.  
> |  
> | This set of helpers is used to get the global pid nr, the virtual (i.e.  
> | seen by task in its namespace) nr and the nr as it is seen from the  
> | specified namespace.  
> |  
> | Signed-off-by: Pavel Emelianov <xemul@openvz.org>  
> |  
> | ---  
> |  
> | include/linux/pid.h | 27 ++++++++  
> | include/linux/sched.h | 108 +++++++++++++++++++++++++++++++++++++-----  
> | kernel/pid.c | 8 +++  
> | 3 files changed, 132 insertions(+), 11 deletions(-)  
> |  
> | diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid.h  
linux-2.6.22-rc4-mm2-2/include/linux/pid.h  
> | --- linux-2.6.22-rc4-mm2.orig/include/linux/pid.h 2007-06-14 12:14:29.000000000 +0400  
> | +++ linux-2.6.22-rc4-mm2-2/include/linux/pid.h 2007-07-04 19:00:38.000000000 +0400  
> | @@ -83,6 +89,9 @@ extern void FASTCALL(detach\_pid(struct t  
> | extern void FASTCALL(transfer\_pid(struct task\_struct \*old,  
> | struct task\_struct \*new, enum pid\_type));  
> |  
> | +struct pid\_namespace;  
> | +extern struct pid\_namespace init\_pid\_ns;  
> | +  
> | /\*  
> | \* look up a PID in the hash table. Must be called with the tasklist\_lock

```

> | * or rcu_read_lock() held.
> | @@ -93,14 +99,36 @@ extern void FASTCALL(detach_pid(struct t
> | extern struct pid *alloc_pid(void);
> | extern void FASTCALL(free_pid(struct pid *pid));
> |
> | +/*
> | + * the helpers to get the pid's id seen from different namespaces
> | + *
> | + * pid_nr() : global id, i.e. the id seen from the init namespace;
> | + * pid_vnr() : virtual id, i.e. the id seen from the namespace this pid
> | + *
> | + * belongs to. this only makes sence when called in the
> | + *
> | + * context of the task that belongs to the same namespace;
> | + * pid_nr_ns() : id seen from the ns specified.
> | + *
> | + * see also task_xid_nr() etc in include/linux/sched.h
> | + */
>
> I think its a bit confusing and error-prone to have both pid_nr() and pid_vnr().
>
> BTW, shouldn't you use pid_vnr() in do_task_stat() ? You currently use pid_nr()

```

Hm... do\_task\_stat() has to use pid\_nr\_ns() actually... I was sure I fixed it in the 15th patch! Let me see...

Yup! It is there:

```

- task->pid,
+ task_pid_nr_ns(task, current->nsproxy->pid_ns),

```

:)

> and that returns the init-pid-ns id right ?

```

>
>
> | +
> | static inline pid_t pid_nr(struct pid *pid)
> | {
> | pid_t nr = 0;
> | if (pid)
> | - nr = pid->nr;
> | + nr = pid->numbers[0].nr;
> | return nr;
> | }
> |
> | +pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns);
> | +
> | +static inline pid_t pid_vnr(struct pid *pid)
> | +{

```

```

> | + pid_t nr = 0;
> | + if (pid)
> | + nr = pid->numbers[pid->level].nr;
> | + return nr;
> | +}
> | +
> | #define do_each_pid_task(pid, type, task) \
> | do { \
> | struct hlist_node *pos___; \
> | diff -upr linux-2.6.22-rc4-mm2.orig/kernel/pid.c linux-2.6.22-rc4-mm2-2/kernel/pid.c
> | --- linux-2.6.22-rc4-mm2.orig/kernel/pid.c 2007-06-14 12:14:29.000000000 +0400
> | +++ linux-2.6.22-rc4-mm2-2/kernel/pid.c 2007-07-04 19:00:38.000000000 +0400
> | @@ -339,6 +379,14 @@ struct pid *find_get_pid(pid_t nr)
> | return pid;
> | }
> |
> | +pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
> | +{
> | + pid_t nr = 0;
> | + if (pid && ns->level <= pid->level)
> | + nr = pid->numbers[ns->level].nr;
> | + return nr;
> | +}
> | +
> | /*
> | * Used by proc to find the first pid that is greater then or equal to nr.
> | *
> | diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/sched.h
linux-2.6.22-rc4-mm2-2/include/linux/sched.h
> | --- linux-2.6.22-rc4-mm2.orig/include/linux/sched.h 2007-07-04 19:00:38.000000000 +0400
> | +++ linux-2.6.22-rc4-mm2-2/include/linux/sched.h 2007-07-04 19:00:38.000000000 +0400
> | @@ -1153,16 +1154,6 @@ struct task_struct {
> | #endif
> | };
> |
> | -static inline pid_t task_pgrp_nr(struct task_struct *tsk)
> | -{
> | - return tsk->signal->pgrp;
> | -}
> | -
> | -static inline pid_t task_session_nr(struct task_struct *tsk)
> | -{
> | - return tsk->signal->__session;
> | -}
> | -
> | static inline void set_task_session(struct task_struct *tsk, pid_t session)
> | {
> | tsk->signal->__session = session;

```

```

> | @@ -1188,6 +1179,104 @@ static inline struct pid *task_session(s
> | return task->group_leader->pids[PIDTYPE_SID].pid;
> | }
> |
> | +struct pid_namespace;
> | +
> | +/*
> | + * the helpers to get the task's different pids as they are seen
> | + * from various namespaces
> | + *
> | + * task_xid_nr() : global id, i.e. the id seen from the init namespace;
> | + * task_xid_vnr() : virtual id, i.e. the id seen from the namespace the task
> | + * belongs to. this only makes sence when called in the
> | + * context of the task that belongs to the same namespace;
> | + * task_xid_nr_ns() : id seen from the ns specified;
> | + *
> | + * set_task_vxid() : assigns a virtual id to a task;
> | + *
> | + * task_ppid_nr_ns() : the parent's id as seen from the namespace specified.
> | + * the result depends on the namespace and whether the
> | + * task in question is the namespace's init. e.g. for the
> | + * namespace's init this will return 0 when called from
> | + * the namespace of this init, or appropriate id otherwise.
> | + *
> | + *
> | + * see also pid_nr() etc in include/linux/pid.h
> | + */
> | +
> | +static inline pid_t task_pid_nr(struct task_struct *tsk)
> | +{
> | + return tsk->pid;
> | +}
> | +
> | +static inline pid_t task_pid_nr_ns(struct task_struct *tsk,
> | + struct pid_namespace *ns)
> | +{
> | + return pid_nr_ns(task_pid(tsk), ns);
> | +}
> | +
> | +static inline pid_t task_pid_vnr(struct task_struct *tsk)
> | +{
> | + return pid_vnr(task_pid(tsk));
> | +}
> | +
> | +
> | +static inline pid_t task_tgid_nr(struct task_struct *tsk)
> | +{
> | + return tsk->tgid;

```



```

> | +}
> | +
> | +static inline pid_t task_tgid_nr_ns(struct task_struct *tsk,
> | + struct pid_namespace *ns)
> | +{
> | + return pid_nr_ns(task_tgid(tsk), ns);
> | +}
> | +
> | +static inline pid_t task_tgid_vnr(struct task_struct *tsk)
> | +{
> | + return pid_vnr(task_tgid(tsk));
> | +}
> | +
> | +
> | +static inline pid_t task_pgrp_nr(struct task_struct *tsk)
> | +{
> | + return tsk->signal->pgrp;
> | +}
> | +
> | +static inline pid_t task_pgrp_nr_ns(struct task_struct *tsk,
> | + struct pid_namespace *ns)
> | +{
> | + return pid_nr_ns(task_pgrp(tsk), ns);
> | +}
> | +
> | +static inline pid_t task_pgrp_vnr(struct task_struct *tsk)
> | +{
> | + return pid_vnr(task_pgrp(tsk));
> | +}
> | +
> | +
> | +static inline pid_t task_session_nr(struct task_struct *tsk)
> | +{
> | + return tsk->signal->__session;
> | +}
> | +
> | +static inline pid_t task_session_nr_ns(struct task_struct *tsk,
> | + struct pid_namespace *ns)
> | +{
> | + return pid_nr_ns(task_session(tsk), ns);
> | +}
> | +
> | +static inline pid_t task_session_vnr(struct task_struct *tsk)
> | +{
> | + return pid_vnr(task_session(tsk));
> | +}
> | +
> | +

```

```
> | +static inline pid_t task_ppid_nr_ns(struct task_struct *tsk,  
> | + struct pid_namespace *ns)  
> | +{  
> | + return pid_nr_ns(task_pid(rcu_dereference(tsk->real_parent)), ns);  
> | +}  
> | +  
> | /**  
> | * pid_alive - check that a task structure is not stale  
> | * @p: Task structure to be checked.  
>
```

---

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

---

---

Subject: Re: [PATCH 8/16] Masquerade the signinfo when sending a pid to a foreign namespace  
Posted by [Pavel Emelianov](#) on Tue, 10 Jul 2007 06:56:14 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

sukadev@us.ibm.com wrote:

```
> Pavel Emelianov [xemul@openvz.org] wrote:  
> | When user send signal from (say) init namespace to any task in a sub  
> | namespace the signinfo struct must not carry the sender's pid value, as  
> | this value may refer to some task in the destination namespace and thus  
> | may confuse the application.  
>  
> Also, do you prevent signals to the child reaper of a container from within  
> its container ? If so, can you show me where you handle it ? I can't  
> seem to find it.  
>  
> And I guess you do allow signals to the child-reaper of a container from  
> its parent container.
```

See my comment below.

```
> |  
> | The consensus was to pretend in this case as if it is the kernel who  
> | sends the signal.  
> |  
> | The pid_ns_accessible() call is introduced to check this pid-to-ns  
> | accessibility.  
> |  
> | Signed-off-by: Pavel Emelianov <xemul@openvz.org>  
> |  
> | ---
```

```

> |
> | include/linux/pid.h | 10 ++++++++
> | kernel/signal.c | 34 ++++++-----
> | 2 files changed, 38 insertions(+), 6 deletions(-)
> |
> | diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid.h
linux-2.6.22-rc4-mm2-2/include/linux/pid.h
> | --- linux-2.6.22-rc4-mm2.orig/include/linux/pid.h 2007-06-14 12:14:29.000000000 +0400
> | +++ linux-2.6.22-rc4-mm2-2/include/linux/pid.h 2007-07-04 19:00:38.000000000 +0400
> | @@ -83,6 +89,16 @@ extern void FASTCALL(detach_pid(struct t
> | return nr;
> | }
> |
> | +/*
> | + * checks whether the pid actually lives in the namespace ns, i.e. it was
> | + * created in this namespace or it was moved there.
> | + */
> | +
> | +static inline int pid_ns_accessible(struct pid_namespace *ns, struct pid *pid)
> | +{
> | + return pid->numbers[pid->level].ns == ns;
> | +}
> | +
> | #define do_each_pid_task(pid, type, task) \
> | do { \
> | struct hlist_node *pos___; \
> | diff -upr linux-2.6.22-rc4-mm2.orig/kernel/signal.c linux-2.6.22-rc4-mm2-2/kernel/signal.c
> | --- linux-2.6.22-rc4-mm2.orig/kernel/signal.c 2007-07-04 19:00:38.000000000 +0400
> | +++ linux-2.6.22-rc4-mm2-2/kernel/signal.c 2007-07-04 19:00:38.000000000 +0400
> | @@ -1124,13 +1124,31 @@ EXPORT_SYMBOL_GPL(kill_pid_info_as_uid);
> | * is probably wrong. Should make it like BSD or SYSV.
> | */
> |
> | -static int kill_something_info(int sig, struct siginfo *info, int pid)
> | +static inline void masquerade_siginfo(struct pid_namespace *src_ns,
> | + struct pid *tgt_pid, struct siginfo *info)
> | +{
> | + if (tgt_pid != NULL && !pid_ns_accessible(src_ns, tgt_pid)) {
> | + /*
> | + * current namespace is not seen from the taks we
> | + * want to send the signal to, so pretend as if it
> | + * is the kernel who does this to avoid pid messing
> | + * by the target
> | + */
> | +
> | + info->si_pid = 0;
> | + info->si_code = SI_KERNEL;
> | + }

```

```

> | +}
> | +
> | +static int kill_something_info(int sig, struct siginfo *info, int pid_nr)
> | {
> |   int ret;
> |   struct pid *pid;
> |   +
> |   rcu_read_lock();
> |   - if (!pid) {
> |   + if (!pid_nr) {
> |     ret = kill_pgrp_info(sig, info, task_pgrp(current));
> |   - } else if (pid == -1) {
> |   + } else if (pid_nr == -1) {
> |     int retval = 0, count = 0;
> |     struct task_struct * p;
>
> So what happens if we run "kill -s <sig> -1" from within a container ?
> Do you terminate all processes in the system or just the process in
> the container ?

```

That's the biggest problem in the whole set. I do not allow for any signal to the namespaces init (and use "standart" init in my experiences), since I have no ideas of how to make it look good.

Checking for abilities in the `sys_kill()` is a solution, but why wasn't it such in the global init case? Why init checks for signals in `get_signal_to_deliver()`. I have to think a bit more with this place. Maybe checking for permissions in `sys_kill` is a good solution.

One of the ideas I had is that the namespace's init has to accept all the signals with `si_code == SI_KERNEL` (this will include signals from parent namespaces as well), but the problem is that struct `siginfo`'s do not reach the `get_signal_to_deliver` in 100% times. If we just could somehow push the `siginfo` to init, I would concern the problem to be solved.

Thanks,  
Pavel

```

> -
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/
>

```

---

Containers mailing list

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Pavel Emelianov](#) on Tue, 10 Jul 2007 06:59:35 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Cedric Le Goater wrote:

>>>> 3. Suka assumes that pid namespace can live without proc mount  
>>>> and tries to make the code work with pid\_ns->proc\_mnt change  
>>>> from NULL to not-NULL from times to times.  
>>>> My code calls the kern\_mount() at the namespace creation and  
>>>> thus the pid\_namespace always works with proc.  
>>>> shouldn't that be done by userspace instead?  
>>> It can be. But when the namespace is being created there's no  
>>> any userspace in it yet.  
>> I'm not talking about the 'userspace inside the space'  
>> I'm talking about the userspace creating the space  
>> (what if I do not want to have any proc mount?)  
>  
> yes, can't we let the user doing the unshare or clone decide whether  
> it needs to mount /proc or not in the new pid namespace ?  
>  
> that's already optional on the host.

If we admit that the pid\_ns->proc\_mnt pointer can change from NULL to some valid mount, then we should track all the possible races with its dereference in the code. We have already caught one of such "tricky" places with Suka's patches and sent the OOPs to him.

My opinion is that having kern\_mount-ed proc makes no discomfort but protects from all the possible races with this pointer dereferencing.

Thanks,  
Pavel

> C.  
> -  
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in  
> the body of a message to [majordomo@vger.kernel.org](mailto:majordomo@vger.kernel.org)  
> More majordomo info at <http://vger.kernel.org/majordomo-info.html>  
> Please read the FAQ at <http://www.tux.org/lkml/>  
>

---

Containers mailing list  
[Containers@lists.linux-foundation.org](mailto:Containers@lists.linux-foundation.org)

---

Subject: Re: [PATCH 0/16] Pid namespaces

Posted by [Pavel Emelianov](#) on Tue, 10 Jul 2007 07:02:51 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

sukadev@us.ibm.com wrote:

> I am not able to find a specific patch that this might be in,  
> but what happens when the child-reaper of a container exits ?

The init namespace's init becomes this init's namespace's init :)

In other words:

```
if (unlikely(tsk == child_reaper(tsk))) {  
    if (tsk->nsproxy->pid_ns != &init_pid_ns)  
        tsk->nsproxy->pid_ns->child_reaper = init_pid_ns.child_reaper;  
    else  
        panic("Attempted to kill init!");  
}
```

> Do you terminate all processes in the container ? I thought  
> that was discussed earlier and the consensus was to terminate  
> all processes in that container and its subordinate containers.  
>  
> Is that not the case now ?

That's the case, but this code works without this change. We can do it later.

> Suka

>

> Pavel Emelianov [[xemul@openvz.org](mailto:xemul@openvz.org)] wrote:

> | This is "submission for inclusion" of hierarchical, not kconfig  
> | configurable, zero overheaded ;) pid namespaces.

> |

> | The overall idea is the following:

> |

> | The namespace are organized as a tree - once a task is cloned  
> | with CLONE\_NEWPIDS (yes, I've also switched to it :) the new  
> | namespace becomes the parent's child and tasks living in the  
> | parent namespace see the tasks from the new one. The numerical  
> | ids are used on the kernel-user boundary, i.e. when we export  
> | pid to user we show the id, that should be used to address the  
> | task in question from the namespace we're exporting this id to.

> |

> | The main difference from Suka's patches are the following:

> |

> | 0. Suka's patches change the kernel/pid.c code too heavy.  
> | This set keeps the kernel code look like it was without  
> | the patches. However, this is a minor issue. The major is:  
> |  
> | 1. Suka's approach is to remove the notion of the task's  
> | numerical pid from the kernel at all. The numbers are  
> | used on the kernel-user boundary or within the kernel but  
> | with the namespace this nr belongs to. This results in  
> | massive changes of struct's members fro int pid to struct  
> | pid \*pid, task->pid becomes the virtual id and so on and  
> | so forth.  
> | My approach is to keep the good old logic in the kernel.  
> | The task->pid is a global and unique pid, find\_pid() finds  
> | the pid by its global id and so on. The virtual ids appear  
> | on the user-kernel boundary only. Thus drivers and other  
> | kernel code may still be unaware of pids unless they do not  
> | communicate with the userspace and get/put numerical pids.  
> |  
> | And some more minor differences:  
> |  
> | 2. Suka's patches have the limit of pid namespace nesting.  
> | My patches do not.  
> |  
> | 3. Suka assumes that pid namespace can live without proc mount  
> | and tries to make the code work with pid\_ns->proc\_mnt change  
> | from NULL to not-NULL from times to times.  
> | My code calls the kern\_mount() at the namespace creation and  
> | thus the pid\_namespace always works with proc.  
> |  
> | There are some small issues that I can describe if someone is  
> | interested.  
> |  
> | The tests like nptl perf, unixbench spawn, getpid and others  
> | didn't reveal any performance degradation in init\_namespace  
> | with the RHEL5 kernel .config file. I admit, that different  
> | .config-s may show that patches hurt the performance, but the  
> | intention was \*not\* to make the kernel work worse with popular  
> | distributions.  
> |  
> | This set has some ways to move forward, but this is some kind  
> | of a core, that do not change the init\_pid\_namespace behavior  
> | (checked with LTP tests) and may require some hacking to do  
> | with the namespaces only.  
> |  
> | Patches apply to 2.6.22-rc6-mm1.  
> -  
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in  
> the body of a message to majordomo@vger.kernel.org

> More majordomo info at <http://vger.kernel.org/majordomo-info.html>  
> Please read the FAQ at <http://www.tux.org/lkml/>  
>

---

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

---

---

Subject: Re: [PATCH 4/16] Change data structures for pid namespaces  
Posted by [Pavel Emelianov](#) on Tue, 10 Jul 2007 07:04:09 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

sukadev@us.ibm.com wrote:

> Cedric Le Goater [[clg@fr.ibm.com](mailto:clg@fr.ibm.com)] wrote:

> | Pavel Emelianov wrote:

> | > struct pid\_namespace will have the kmem\_cache to allocate

> | > the pids from, the parent, as they are hierarchical, and

> | > the level of nesting value.

> | >

> | > struct pid will have a variable length array of pid\_number-s

> | > one for each namespace this pid lives in. The level value

> | > shows the level of the namespace this pid lives in and thus -

> | > the number of elements in the numbers array.

> | >

> | > Signed-off-by: Pavel Emelianov <[xemul@openvz.org](mailto:xemul@openvz.org)>

> | >

> | > ---

> | >

> | > include/linux/init\_task.h | 6 ++++++

> | > include/linux/pid.h | 9 ++++++++

> | > include/linux/pid\_namespace.h | 3 +++

> | > kernel/pid.c | 3 ++-

> | > 4 files changed, 20 insertions(+), 1 deletion(-)

> | >

> | > diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid.h

linux-2.6.22-rc4-mm2-2/include/linux/pid.h

> | > --- linux-2.6.22-rc4-mm2.orig/include/linux/pid.h 2007-06-14 12:14:29.000000000 +0400

> | > +++ linux-2.6.22-rc4-mm2-2/include/linux/pid.h 2007-07-04 19:00:38.000000000 +0400

> | > @@ -40,6 +40,13 @@ enum pid\_type

> | > \* processes.

> | > \*/

> | >

> | > +struct pid\_number {

> | > + /\* Try to keep pid\_chain in the same cacheline as nr for find\_pid \*/

> | > + int nr;

> | > + struct pid\_namespace \*ns;



```

> | > + struct hlist_node pid_chain;
> | > +};
>
> We meant to go back and look at removing the extra 'struct pid *' we had
> here. Looks like you did that. Cool.
>
> | > +
> | > struct pid
> | > {
> | > atomic_t count;
> | > @@ -40,6 +40,8 @@ enum pid_type
> | > /* lists of tasks that use this pid */
> | > struct hlist_head tasks[PIDTYPE_MAX];
> | > struct rcu_head rcu;
> | > + int level;
> | > + struct pid_number numbers[1];
> | > };
> | >
> | > extern struct pid init_struct_pid;
> | > diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid_namespace.h
linux-2.6.22-rc4-mm2-2/include/linux/pid_namespace.h
> | > --- linux-2.6.22-rc4-mm2.orig/include/linux/pid_namespace.h 2007-06-14
12:14:29.000000000 +0400
> | > +++ linux-2.6.22-rc4-mm2-2/include/linux/pid_namespace.h 2007-07-04 19:00:39.000000000
+0400
> | > @@ -16,7 +15,10 @@ struct pidmap {
> | > struct kref kref;
> | > struct pidmap pidmap[PIDMAP_ENTRIES];
> | > int last_pid;
> | > + int level;
> | > struct task_struct *child_reaper;
> | > + struct kmem_cache *pid_cache;
> |
> | so, that looks like a good idea to have the cache in the pidmap. could you
> | push that independently to see how it all fits together ?
>
> Yes. I like this idea too.

```

OK. I will collect the patches you Aacked and send them to Andrew independently.

```

> |
> | thanks,
> |
> | C.
>

```

---

Containers mailing list

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Pavel Emelianov](#) on Tue, 10 Jul 2007 07:05:18 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Badari Pulavarty wrote:

> On Mon, 2007-07-09 at 22:06 +0200, Cedric Le Goater wrote:  
>> Badari Pulavarty wrote:  
>>> On Fri, 2007-07-06 at 12:01 +0400, Pavel Emelianov wrote:  
>>>> This is "submission for inclusion" of hierarchical, not kconfig  
>>>> configurable, zero overheaded ;) pid namespaces.  
>>> Not able to boot my ppc64 machine with the patchset :(  
>> I can't boot either on a x86\_64 but I don't even have logs to send :(  
>  
> Yes. It blew up way early in the boot on my x86\_64, so nothing came  
> up on the console to capture (blank screen) :(

That's bad news... I will re-check for it. It seems to me that  
I lost some parts during the patch splitting... :(

> Thanks,  
> Badari  
>  
>

---

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

---

---

Subject: Re: [PATCH 1/16] Round up the API  
Posted by [akpm](#) on Tue, 10 Jul 2007 07:34:50 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 10 Jul 2007 10:40:13 +0400 Pavel Emelianov <xemul@openvz.org> wrote:

> > I think we are all ok with it. right ?  
>  
> Right. That's already the 3rd time I send it to Andrew...

I'm basically ignoring all the containers/resource-control stuff, waiting  
for it to appear to have settled down. It's quite unclear which patches  
are at the RFC stage and which are at the ready-to-go stage.

If you guys could gather and maintain the acked-by's and make it clear what the maturity level is on each patch series it would help, thanks.

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Pavel Emelianov](#) on Tue, 10 Jul 2007 09:41:16 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

sukadev@us.ibm.com wrote:

> Pavel Emelianov [xemul@openvz.org] wrote:

> | This is "submission for inclusion" of hierarchical, not kconfig  
> | configurable, zero overheaded ;) pid namespaces.

> |  
> | The overall idea is the following:

> |  
> | The namespace are organized as a tree - once a task is cloned  
> | with CLONE\_NEWPIDS (yes, I've also switched to it :) the new

>  
> Can you really clone() a pid namespace all by itself ?  
> copy\_namespaces() has the following:

```
>  
>  
>     if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |  
CLONE_NEWUSER)))  
>         return 0;
```

>  
> doesn't it mean you cannot create a pid namespace using clone() unless  
> one of the above flags are also specified ?

>  
> unshare\_nsproxy\_namespaces() has the following correct check:

```
>  
>     if (!(unshare_flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |  
>         CLONE_NEWUSER | CLONE_NEWPIDS)))  
>         return 0;
```

I have already pointed this out. I attached the previous version of this patch where I cloned the namespaces via unshare :( However the copy\_pid\_ns code looks correct. I will resend it altogether.

My bad. I have to stop working at times I want to sleep...

> BTW, why not use CLONE\_NEWPID and drop the 'S' ? We don't have 'S' with

> other namespaces.

CLONE\_NEWPID? Hm... I see no difference. OK I'll switch to it.

> Suka  
>

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Pavel Emelianov](#) on Tue, 10 Jul 2007 11:30:16 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Cedric Le Goater wrote:  
> Badari Pulavarty wrote:  
>> On Fri, 2007-07-06 at 12:01 +0400, Pavel Emelianov wrote:  
>>> This is "submission for inclusion" of hierarchical, not kconfig  
>>> configurable, zero overheaded ;) pid namespaces.  
>> Not able to boot my ppc64 machine with the patchset :(  
>  
> I can't boot either on a x86\_64 but I don't even have logs to send :(

Neither can I. And I cannot boot clean 2.6.22-rc6-mm1 either :(  
Does someone already know what the reason is?

> C.  
>

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Daniel Lezcano](#) on Tue, 10 Jul 2007 12:05:08 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Pavel Emelianov wrote:  
> Cedric Le Goater wrote:  
>> Badari Pulavarty wrote:  
>>> On Fri, 2007-07-06 at 12:01 +0400, Pavel Emelianov wrote:  
>>>> This is "submission for inclusion" of hierarchical, not kconfig

>>>> configurable, zero overheaded ;) pid namespaces.  
>>> Not able to boot my ppc64 machine with the patchset :(  
>> I can't boot either on a x86\_64 but I don't even have logs to send :(

Did you try to append at the kernel command line the parameter:

```
earlyprintk=serial,ttys0,keep
```

>  
> Neither can I. And I cannot boot clean 2.6.22-rc6-mm1 either :(  
> Does someone already know what the reason is?

>  
>> C.

>>

>

>

---

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

---

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

---

Subject: Re: [PATCH 4/16] Change data structures for pid namespaces

Posted by [Cedric Le Goater](#) on Tue, 10 Jul 2007 12:07:45 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

```
>> | > +
>> | > struct pid
>> | > {
>> | > atomic_t count;
>> | > @@ -40,6 +40,8 @@ enum pid_type
>> | > /* lists of tasks that use this pid */
>> | > struct hlist_head tasks[PIDTYPE_MAX];
>> | > struct rcu_head rcu;
>> | > + int level;
>> | > + struct pid_number numbers[1];
>> | > };
>> | >
>> | > extern struct pid init_struct_pid;
>> | > diff -upr linux-2.6.22-rc4-mm2.orig/include/linux/pid_namespace.h
linux-2.6.22-rc4-mm2-2/include/linux/pid_namespace.h
>> | > --- linux-2.6.22-rc4-mm2.orig/include/linux/pid_namespace.h 2007-06-14
12:14:29.000000000 +0400
>> | > +++ linux-2.6.22-rc4-mm2-2/include/linux/pid_namespace.h 2007-07-04
```

19:00:39.000000000 +0400

```
>> | > @@ -16,7 +15,10 @@ struct pidmap {
>> | > struct kref kref;
>> | > struct pidmap pidmap[PIDMAP_ENTRIES];
>> | > int last_pid;
>> | > + int level;
>> | > struct task_struct *child_reaper;
>> | > + struct kmem_cache *pid_cache;
>> |
>> | so, that looks like a good idea to have the cache in the pidmap. could you
>> | push that independently to see how it all fits together ?
>>
>> Yes. I like this idea too.
>
> OK. I will collect the patches you Aacked and send them to Andrew independently.
```

good thanks,

however that part on the \*pid\_cache will require some rework on your patchset.

C.

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Pavel Emelianov](#) on Tue, 10 Jul 2007 13:03:13 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Daniel Lezcano wrote:

```
> Pavel Emelianov wrote:
>> Cedric Le Goater wrote:
>>> Badari Pulavarty wrote:
>>>> On Fri, 2007-07-06 at 12:01 +0400, Pavel Emelianov wrote:
>>>>> This is "submission for inclusion" of hierarchical, not kconfig
>>>>> configurable, zero overheaded ;) pid namespaces.
>>>> Not able to boot my ppc64 machine with the patchset :(
>>> I can't boot either on a x86_64 but I don't even have logs to send :(
>
> Did you tried to append at the kernel command line the parameter:
>
> earlyprintk=serial,ttys0,keep
```

I have already faced the problem, thanks :)

The bad-guy is CONFIG\_DMAR. I'll talk to intel developers about this :)

And a couple of patches to fix proc (I'll send them in a moment). i386 works without is and all the others crash. Kirill explained me why. It looks like a big problem for i386. He told, that during the boot first 32MiBs of memory are mapped and thus all NULL pointer dereferences succeed. Is it good?

Thanks,  
Pavel

```
>>
>> Neither can I. And I cannot boot clean 2.6.22-rc6-mm1 either :(
>> Does someone already know what the reason is?
>>
>>> C.
>>>
>>
>> _____
>> Containers mailing list
>> Containers@lists.linux-foundation.org
>> https://lists.linux-foundation.org/mailman/listinfo/containers
>
> -
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/
>
```

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Pavel Emelianov](#) on Tue, 10 Jul 2007 13:06:08 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

```
> Not able to boot my ppc64 machine with the patchset :(
>
> Thanks,
> Badari
```

That's the hunk lost during the split:

```
--- ./fs/proc/root.c.procfix 2007-07-10 13:52:08.000000000 +0400
+++ ./fs/proc/root.c 2007-07-10 15:23:20.000000000 +0400
@@ -111,7 +111,7 @@ void __init proc_root_init(void)
```

```
err = register_filesystem(&proc_fs_type);
if (err)
    return;
- proc_mnt = kern_mount(&proc_fs_type);
+ proc_mnt = kern_mount_data(&proc_fs_type, &init_pid_ns);
err = PTR_ERR(proc_mnt);
if (IS_ERR(proc_mnt)) {
    unregister_filesystem(&proc_fs_type);
```

With this machine should boot fine.

Thanks,  
Pavel

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Pavel Emelianov](#) on Tue, 10 Jul 2007 13:08:42 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

sukadev@us.ibm.com wrote:

- > Pavel Emelianov [xemul@openvz.org] wrote:
- > | This is "submission for inclusion" of hierarchical, not kconfig
- > | configurable, zero overheaded ;) pid namespaces.
- > |
- > | The overall idea is the following:
- > |
- > | The namespace are organized as a tree - once a task is cloned
- > | with CLONE\_NEWPIDS (yes, I've also switched to it :) the new
- > |
- > Can you really clone() a pid namespace all by itself ?
- > copy\_namespaces() has the following:
- > |
- > |
- > | if (!(flags & (CLONE\_NEWNS | CLONE\_NEWUTS | CLONE\_NEWIPC |
- > CLONE\_NEWUSER)))
- > | return 0;
- > |
- > doesn't it mean you cannot create a pid namespace using clone() unless
- > one of the above flags are also specified ?
- > |
- > unshare\_nsproxy\_namespaces() has the following correct check:
- > |
- > | if (!(unshare\_flags & (CLONE\_NEWNS | CLONE\_NEWUTS | CLONE\_NEWIPC |



```

>         CLONE_NEWUSER | CLONE_NEWPIDS)))
>         return 0;

```

I have lost a couple of hunks when I splitted the patch :(  
That's the correct version, cap\_set fix and the renamed CLONE\_ flag.

```

--- ./include/linux/sched.h.fix 2007-07-06 11:09:33.000000000 +0400
+++ ./include/linux/sched.h 2007-07-10 13:48:19.000000000 +0400
@@ -26,7 +26,7 @@
#define CLONE_NEWUTS 0x04000000 /* New utsname group? */
#define CLONE_NEWIPC 0x08000000 /* New ipcs */
#define CLONE_NEWUSER 0x10000000 /* New user namespace */
-#define CLONE_NEWPIDS 0x20000000 /* New pids */
+#define CLONE_NEWPID 0x20000000 /* New pids */

/*
 * Scheduling policies
--- ./kernel/capability.c.fix 2007-07-06 11:09:33.000000000 +0400
+++ ./kernel/capability.c 2007-07-10 13:50:16.000000000 +0400
@@ -103,7 +103,7 @@ static inline int cap_set_pg(int pgrp_nr
int found = 0;
struct pid *pgrp;

- pgrp = find_pid(pgrp_nr);
+ pgrp = find_pid_ns(pgrp_nr, current->nsproxy->pid_ns);
do_each_pid_task(pgrp, PIDTYPE_PGID, g) {
target = g;
while_each_thread(g, target) {
--- ./kernel/fork.c.fix 2007-07-06 11:09:33.000000000 +0400
+++ ./kernel/fork.c 2007-07-10 13:48:13.000000000 +0400
@@ -1267,7 +1267,7 @@ static struct task_struct *copy_process(
__ptrace_link(p, current->parent);

if (thread_group_leader(p)) {
- if (clone_flags & CLONE_NEWPIDS) {
+ if (clone_flags & CLONE_NEWPID) {
p->nsproxy->pid_ns->child_reaper = p;
p->signal->tty = NULL;
p->signal->pgrp = p->pid;
@@ -1434,7 +1434,7 @@ long do_fork(unsigned long clone_flags,
else
p->state = TASK_STOPPED;

- nr = (clone_flags & CLONE_NEWPIDS) ?
+ nr = (clone_flags & CLONE_NEWPID) ?
pid_nr_ns(task_pid(p), current->nsproxy->pid_ns) :
pid_vnr(task_pid(p));

```

```

--- ./kernel/nsproxy.c.fix 2007-07-06 11:09:33.000000000 +0400
+++ ./kernel/nsproxy.c 2007-07-10 13:48:13.000000000 +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)) {
@@ -184,7 +185,7 @@ int unshare_nsproxy_namespaces(unsigned
    int err = 0;

    if (!(unshare_flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
-   CLONE_NEWUSER | CLONE_NEWPIDS)))
+   CLONE_NEWUSER)))
    return 0;

    if (!capable(CAP_SYS_ADMIN))
--- ./kernel/pid.c.fix 2007-07-06 11:09:33.000000000 +0400
+++ ./kernel/pid.c 2007-07-10 13:48:19.000000000 +0400
@@ -523,7 +523,7 @@ struct pid_namespace *copy_pid_ns(unsigned
    BUG_ON(!old_ns);
    get_pid_ns(old_ns);
    new_ns = old_ns;
- if (!(flags & CLONE_NEWPIDS))
+ if (!(flags & CLONE_NEWPID))
    goto out;

    new_ns = ERR_PTR(-EINVAL);

```

---

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

---



---

**Subject: Re: [PATCH 0/16] Pid namespaces**  
Posted by [Badari Pulavarty](#) on Tue, 10 Jul 2007 20:33:14 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 2007-07-10 at 17:06 +0400, Pavel Emelianov wrote:  
> > Not able to boot my ppc64 machine with the patchset :(  
> >  
> > Thanks,  
> > Badari  
>

```
> That's the hunk lost during the split:
>
> --- ./fs/proc/root.c.procfix 2007-07-10 13:52:08.000000000 +0400
> +++ ./fs/proc/root.c 2007-07-10 15:23:20.000000000 +0400
> @@ -111,7 +111,7 @@ void __init proc_root_init(void)
> err = register_filesystem(&proc_fs_type);
> if (err)
> return;
> - proc_mnt = kern_mount(&proc_fs_type);
> + proc_mnt = kern_mount_data(&proc_fs_type, &init_pid_ns);
> err = PTR_ERR(proc_mnt);
> if (IS_ERR(proc_mnt)) {
> unregister_filesystem(&proc_fs_type);
>
>
> With this machine should boot fine.
```

Yes. My ppc64 box booted fine with this patch.

Thanks,  
Badari

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Badari Pulavarty](#) on Tue, 10 Jul 2007 20:34:16 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, 2007-07-10 at 15:30 +0400, Pavel Emelianov wrote:  
> Cedric Le Goater wrote:  
> > Badari Pulavarty wrote:  
> >> On Fri, 2007-07-06 at 12:01 +0400, Pavel Emelianov wrote:  
> >>> This is "submission for inclusion" of hierarchical, not kconfig  
> >>> configurable, zero overheaded ;) pid namespaces.  
> >> Not able to boot my ppc64 machine with the patchset :(  
> >  
> > I can't boot either on a x86\_64 but I don't even have logs to send :(  
>  
> Neither can I. And I cannot boot clean 2.6.22-rc6-mm1 either :(  
> Does someone already know what the reason is?

2.6.22-rc6-mm1 boots fine on my x86-64 machine.

your patches with the fix (in fs/proc/root.c), I am able to

boot my x86-64 machine also.

Thanks,  
Badari

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Matt Mackall](#) on Wed, 11 Jul 2007 01:16:52 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Fri, Jul 06, 2007 at 12:01:59PM +0400, Pavel Emelianov wrote:  
> This is "submission for inclusion" of hierarchical, not kconfig  
> configurable, zero overheaded ;) pid namespaces.

How big is it?

Do I want it on my cell phone or on my wireless router?

--

Mathematics is the supreme nostalgia of our time.

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Pavel Emelianov](#) on Wed, 11 Jul 2007 06:39:02 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Matt Mackall wrote:  
> On Fri, Jul 06, 2007 at 12:01:59PM +0400, Pavel Emelianov wrote:  
>> This is "submission for inclusion" of hierarchical, not kconfig  
>> configurable, zero overheaded ;) pid namespaces.  
>  
> How big is it?

Summary: 50 files changed, 877 insertions, 241 deletions. Besides,  
each patch has a diffstat output in the letter.

> Do I want it on my cell phone or on my wireless router?  
>

I do not know, but those who do not need the namespaces at all won't even notice its presence. I tried to do such.

Pavel.

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Matt Mackall](#) on Wed, 11 Jul 2007 15:14:42 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Wed, Jul 11, 2007 at 10:39:02AM +0400, Pavel Emelianov wrote:  
> Matt Mackall wrote:  
> > On Fri, Jul 06, 2007 at 12:01:59PM +0400, Pavel Emelianov wrote:  
> >> This is "submission for inclusion" of hierarchical, not kconfig  
> >> configurable, zero overheaded ;) pid namespaces.  
> >  
> > How big is it?  
>  
> Summary: 50 files changed, 877 insertions, 241 deletions. Besides,  
> each patch has a diffstat output in the letter.

No, how much impact does it have on the compiled kernel's size?

> > Do I want it on my cell phone or on my wireless router?  
>  
> I do not know, but those who do not need the namespaces at  
> all won't even notice its presence. I tried to do such.

It seems likely that something that adds a net 600 lines of code will have a noticeable footprint.

--

Mathematics is the supreme nostalgia of our time.

---

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

---

---

Subject: Re: [PATCH 10/16] Changes in copy\_process() to work with pid namespaces  
Posted by [Sukadev Bhattiprolu](#) on Thu, 12 Jul 2007 00:21:02 GMT

---

Pavel Emelianov [xemul@openvz.org] wrote:

| We must pass the namespace pointer to the alloc\_pid() to  
| show what namespace to allocate the pid from and we should  
| call this \*after\* the namespace is copied.

| Essentially, the task->pid etc initialization is done after  
| the alloc\_pid().

| To do so I move the alloc\_pid() inside copy\_process() and  
| introduce an argument to the alloc\_pid() function.

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

| ---

| include/linux/pid.h | 2 +-  
| kernel/fork.c | 29 ++++++-----  
| kernel/pid.c | 2 +-  
| 3 files changed, 19 insertions(+), 14 deletions(-)

| --- ./include/linux/pid.h.ve9 2007-07-06 11:03:55.000000000 +0400  
| +++ ./include/linux/pid.h 2007-07-06 11:03:55.000000000 +0400  
| @@ -116,7 +116,7 @@ extern struct pid \*FASTCALL(find\_pid\_ns(  
| extern struct pid \*find\_get\_pid(int nr);  
| extern struct pid \*find\_ge\_pid(int nr, struct pid\_namespace \*);

| -extern struct pid \*alloc\_pid(void);  
| +extern struct pid \*alloc\_pid(struct pid\_namespace \*ns);  
| extern void FASTCALL(free\_pid(struct pid \*pid));

| /\*

| --- ./kernel/fork.c.ve9 2007-07-06 11:03:55.000000000 +0400  
| +++ ./kernel/fork.c 2007-07-06 11:04:07.000000000 +0400  
| @@ -50,6 +50,7 @@  
| #include <linux/taskstats\_kern.h>  
| #include <linux/random.h>  
| #include <linux/tty.h>  
| +#include <linux/pid.h>  
|  
| #include <asm/pgtable.h>  
| #include <asm/pgalloc.h>  
| @@ -1032,7 +1033,6 @@ static struct task\_struct \*copy\_process(  
| p->did\_exec = 0;  
| delayacct\_tsk\_init(p); /\* Must remain after dup\_task\_struct() \*/  
| copy\_flags(clone\_flags, p);  
| - p->pid = pid\_nr(pid);  
| INIT\_LIST\_HEAD(&p->children);

```

| INIT_LIST_HEAD(&p->sibling);
| p->vfork_done = NULL;
| @@ -1107,10 +1107,6 @@ static struct task_struct *copy_process(
| p->blocked_on = NULL; /* not blocked yet */
| #endif
|
| - p->tgid = p->pid;
| - if (clone_flags & CLONE_THREAD)
| - p->tgid = current->tgid;
| -
| if ((retval = security_task_alloc(p)))
| goto bad_fork_cleanup_policy;
| if ((retval = audit_alloc(p)))
| @@ -1132,9 +1128,14 @@ static struct task_struct *copy_process(
| goto bad_fork_cleanup_mm;
| if ((retval = copy_namespaces(clone_flags, p)))
| goto bad_fork_cleanup_keys;
| + if (likely(pid == NULL)) {
| + pid = alloc_pid(p->nsproxy->pid_ns);
| + if (pid == NULL)
| + goto bad_fork_cleanup_namespaces;
| + }
| retval = copy_thread(0, clone_flags, stack_start, stack_size, p, regs);
| if (retval)
| - goto bad_fork_cleanup_namespaces;
| + goto bad_fork_cleanup_pid;
|
| p->set_child_tid = (clone_flags & CLONE_CHILD_SETTID) ? child_tidptr : NULL;
| /*
| @@ -1255,6 +1256,11 @@ static struct task_struct *copy_process(
| }
| }
|
| + p->pid = pid_nr(pid);
| + p->tgid = p->pid;
| + if (clone_flags & CLONE_THREAD)
| + p->tgid = current->tgid;
| +
| if (likely(p->pid)) {
| add_parent(p);
| if (unlikely(p->ptrace & PT_PTRACED))
| @@ -1288,6 +1294,8 @@ static struct task_struct *copy_process(
| proc_fork_connector(p);
| return p;
|
| +bad_fork_cleanup_pid:
| + free_pid(pid);

```

If copy\_process() was called from fork\_idle(), pid would refer to &init\_struct\_pid. Don't you need something like:

```
if (pid == &init_struct_pid)
    return;
```

in free\_pid() to not do anything in that case ?

```
| bad_fork_cleanup_namespaces:
|   exit_task_namespaces(p);
| bad_fork_cleanup_keys:
| @@ -1380,19 +1388,16 @@ long do_fork(unsigned long clone_flags,
| {
|   struct task_struct *p;
|   int trace = 0;
| - struct pid *pid = alloc_pid();
|   long nr;
|
| - if (!pid)
| - return -EAGAIN;
| - nr = pid->nr;
|   if (unlikely(current->ptrace)) {
|     trace = fork_traceflag (clone_flags);
|     if (trace)
|       clone_flags |= CLONE_PTRACE;
|   }
|
| - p = copy_process(clone_flags, stack_start, regs, stack_size, parent_tidptr, child_tidptr, pid);
| + p = copy_process(clone_flags, stack_start, regs, stack_size,
| + parent_tidptr, 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.
| @@ -1418,6 +1423,7 @@ long do_fork(unsigned long clone_flags,
|   else
|     p->state = TASK_STOPPED;
|
| + nr = pid_vnr(task_pid(p));
|   if (unlikely (trace)) {
|     current->ptrace_message = nr;
|     ptrace_notify ((trace << 8) | SIGTRAP);
| @@ -1433,7 +1439,6 @@ long do_fork(unsigned long clone_flags,
|   }
| }
| } else {
| - free_pid(pid);
|   nr = PTR_ERR(p);
```



```
| }
| return nr;
| --- ./kernel/pid.c.ve9 2007-07-06 11:03:55.000000000 +0400
| +++ ./kernel/pid.c 2007-07-06 11:03:55.000000000 +0400
| @@ -206,7 +206,7 @@ fastcall void free_pid(struct pid *pid)
|     call_rcu(&pid->rcu, delayed_put_pid);
| }
|
| -struct pid *alloc_pid(void)
| +struct pid *alloc_pid(struct pid_namespace *pid_ns)
| {
|     struct pid *pid;
|     enum pid_type type;
```

---

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>

---

---

Subject: Re: [PATCH 0/16] Pid namespaces

Posted by [Sukadev Bhattiprolu](#) on Thu, 12 Jul 2007 03:19:37 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Pavel Emelianov [xemul@openvz.org] wrote:

| sukadev@us.ibm.com wrote:

| > Pavel Emelianov [xemul@openvz.org] wrote:

| > | This is "submission for inclusion" of hierarchical, not kconfig

| > | configurable, zero overheaded ;) pid namespaces.

| > |

| > | The overall idea is the following:

| > |

| > | The namespace are organized as a tree - once a task is cloned

| > | with CLONE\_NEWPIDS (yes, I've also switched to it :) the new

| > |

| > | Can you really clone() a pid namespace all by itself ?

| > | copy\_namespaces() has the following:

| > |

| > |

| > | if (!(flags & (CLONE\_NEWNS | CLONE\_NEWUTS | CLONE\_NEWIPC |  
CLONE\_NEWUSER)))

| > | return 0;

| > |

| > | doesn't it mean you cannot create a pid namespace using clone() unless

| > | one of the above flags are also specified ?

| > |

| > | unshare\_nsproxy\_namespaces() has the following correct check:

| > |

| > | if (!(unshare\_flags & (CLONE\_NEWNS | CLONE\_NEWUTS | CLONE\_NEWIPC |

```

| >                 CLONE_NEWUSER | CLONE_NEWPIDS)))
| >                 return 0;
|
| I have lost a couple of hunks when I splitted the patch :(
| That's the correct version, cap_set fix and the renamed CLONE_ flag.
|
| --- ./include/linux/sched.h.fix 2007-07-06 11:09:33.000000000 +0400
| +++ ./include/linux/sched.h 2007-07-10 13:48:19.000000000 +0400
| @@ -26,7 +26,7 @@
| #define CLONE_NEWUTS  0x04000000 /* New utsname group? */
| #define CLONE_NEWIPC  0x08000000 /* New ipcns */
| #define CLONE_NEWUSER 0x10000000 /* New user namespace */
| -#define CLONE_NEWPIDS 0x20000000 /* New pids */
| +#define CLONE_NEWPID  0x20000000 /* New pids */
|
| /*
|  * Scheduling policies
| --- ./kernel/capability.c.fix 2007-07-06 11:09:33.000000000 +0400
| +++ ./kernel/capability.c 2007-07-10 13:50:16.000000000 +0400
| @@ -103,7 +103,7 @@ static inline int cap_set_pg(int pgrp_nr
|  int found = 0;
|  struct pid *pgrp;
|
| - pgrp = find_pid(pgrp_nr);
| + pgrp = find_pid_ns(pgrp_nr, current->nsproxy->pid_ns);
|  do_each_pid_task(pgrp, PIDTYPE_PGID, g) {
|    target = g;
|    while_each_thread(g, target) {
| --- ./kernel/fork.c.fix 2007-07-06 11:09:33.000000000 +0400
| +++ ./kernel/fork.c 2007-07-10 13:48:13.000000000 +0400
| @@ -1267,7 +1267,7 @@ static struct task_struct *copy_process(
|  __ptrace_link(p, current->parent);
|
|  if (thread_group_leader(p)) {
| - if (clone_flags & CLONE_NEWPIDS) {
| + if (clone_flags & CLONE_NEWPID) {
|   p->nsproxy->pid_ns->child_reaper = p;
|   p->signal->tty = NULL;
|   p->signal->pgrp = p->pid;
| @@ -1434,7 +1434,7 @@ long do_fork(unsigned long clone_flags,
|  else
|   p->state = TASK_STOPPED;
|
| - nr = (clone_flags & CLONE_NEWPIDS) ?
| + nr = (clone_flags & CLONE_NEWPID) ?
|   pid_nr_ns(task_pid(p), current->nsproxy->pid_ns) :
|   pid_vnr(task_pid(p));

```

```

| --- ./kernel/nsproxy.c.fix 2007-07-06 11:09:33.000000000 +0400
| +++ ./kernel/nsproxy.c 2007-07-10 13:48:13.000000000 +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)) {
| @@ -184,7 +185,7 @@ int unshare_nsproxy_namespaces(unsigned
|   int err = 0;
|
|   if (!(unshare_flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
| - CLONE_NEWUSER | CLONE_NEWPIDS)))
| + CLONE_NEWUSER)))
|     return 0;
|
|   if (!capable(CAP_SYS_ADMIN))
| --- ./kernel/pid.c.fix 2007-07-06 11:09:33.000000000 +0400
| +++ ./kernel/pid.c 2007-07-10 13:48:19.000000000 +0400
| @@ -523,7 +523,7 @@ struct pid_namespace *copy_pid_ns(unsigned
|   BUG_ON(!old_ns);
|   get_pid_ns(old_ns);
|   new_ns = old_ns;
| - if (!(flags & CLONE_NEWPIDS))
| + if (!(flags & CLONE_NEWPID))
|     goto out;
|
|   new_ns = ERR_PTR(-EINVAL);

```

I applied the above patch and the following patch to your 16 patches you first sent.

```

--- ./fs/proc/root.c.procfix 2007-07-10 13:52:08.000000000 +0400
+++ ./fs/proc/root.c 2007-07-10 15:23:20.000000000 +0400
@@ -111,7 +111,7 @@ void __init proc_root_init(void)
    err = register_filesystem(&proc_fs_type);
    if (err)
        return;
-   proc_mnt = kern_mount(&proc_fs_type);
+   proc_mnt = kern_mount_data(&proc_fs_type, &init_pid_ns);

```

My x86\_64 system boots fine but crashes as below, when I run my 'pidns\_exec' test with a simple program that prints getpid(), getppid()

etc of the process in the child pid ns.

Pls see

<http://www.geocities.com/sukadevb/Pidspace/2.6.22-rc6-mm1-pavel-1.tgz>

for the patches I currently have applied and let me know if I need more on top.

And see

<http://www.geocities.com/sukadevb/Pidspace/Test1/>

for the test programs. You may need to run the 'mypid-loop.x' script to repro the crash. The pidns\_exec.c program calls clone() with CLONE\_NEWPID and execs the given program (it was included in Patch 0 of the patchset I posted to Containers).

Suka

```
login: Unable to handle kernel NULL pointer dereference at 00000000000002fc RIP:
[<ffffffff802b9e5e>] proc_get_sb+0xfb/0x138
PGD 104d53067 PUD 104d4d067 PMD 0
Oops: 0002 [1] SMP
CPU 2
Modules linked in:
Pid: 3279, comm: pidns_exec Not tainted 2.6.22-rc6-mm1-ovz1 #10
RIP: 0010:[<ffffffff802b9e5e>] [<ffffffff802b9e5e>] proc_get_sb+0xfb/0x138
RSP: 0018:ffff8101029d7d28 EFLAGS: 00010202
RAX: ffff810100651840 RBX: ffff810104461400 RCX: ffff810100651878
RDX: 0000000000000000 RSI: ffffffff806e5460 RDI: 0000000000000238
RBP: ffff810102d886f8 R08: ffff810104461400 R09: ffff810100026000
R10: 0000000000000000 R11: 0000000000000002 R12: ffff8101029c6000
R13: 0000000002000000 R14: ffffffff806ee920 R15: ffff810102088cc0
FS: 00002b0b499ec6f0(0000) GS:ffff81010069c3c0(0000) knlGS:0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 000000008005003b
CR2: 00000000000002fc CR3: 000000010381b000 CR4: 000000000000006e0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 00000000ffff0ff0 DR7: 0000000000000400
Process pidns_exec (pid: 3279, threadinfo ffff8101029d6000, task ffff81010269e7f0)
Stack: ffff810102088cc0 ffff810102088cc0 00000000ffffff4 ffffffff806ee920
ffff810065f9d9 ffff8101029c6000 0000000002000000 ffffffff80287164
00000000000000d0 ffff8101029c6000 ffffffff806e5460 ffff8101029c6000
Call Trace:
[<ffffffff80287164>] vfs_kern_mount+0x4f/0x8b
[<ffffffff802b9cf4>] pid_ns_prepare_proc+0x13/0x2e
[<ffffffff80245be3>] copy_pid_ns+0xd7/0x164
[<ffffffff8024af34>] create_new_namespaces+0xde/0x192
```

[<ffffff8024b0aa>] copy\_namespaces+0x4b/0x85  
[<ffffff802347e2>] copy\_process+0xcb4/0x1439  
[<ffffff8020bbee>] system\_call+0x7e/0x83  
[<ffffff8023556a>] do\_fork+0x6c/0x1e7  
[<ffffff8020bf07>] ptregscall\_common+0x67/0xb0

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Badari Pulavarty](#) on Thu, 12 Jul 2007 18:55:59 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Wed, 2007-07-11 at 20:19 -0700, sukadev@us.ibm.com wrote:  
> Pavel Emelianov [xemul@openvz.org] wrote:  
> | sukadev@us.ibm.com wrote:  
> |..  
> My x86\_64 system boots fine but crashes as below, when I run my  
> 'pidns\_exec' test with a simple program that prints getpid(), getppid()  
> etc of the process in the child pid ns.  
>  
> Pls see  
>  
> <http://www.geocities.com/sukadevb/Pidspace/2.6.22-rc6-mm1-pavel-1.tgz>  
>  
> for the patches I currently have applied and let me know if I need more  
> on top.  
>  
> And see  
>  
> <http://www.geocities.com/sukadevb/Pidspace/Test1/>  
>  
> for the test programs. You may need to run the 'mypid-loop.x' script  
> to repro the crash. The pidns\_exec.c program calls clone() with CLONE\_NEWPID  
> and execs the given program (it was included in Patch 0 of the patchset I  
> posted to Containers).  
>  
> Suka  
>  
> login: Unable to handle kernel NULL pointer dereference at 00000000000002fc RIP:

Yes. I am noticing different set of problems on my ppc64 with above tests and patchset.

Thanks,  
Badari

Not cloning container for unused subsystem ns  
Not cloning container for unused subsystem ns  
Not cloning container for unused subsystem ns  
Not cloning container for unused subsystem ns  
Not cloning container for unused subsystem ns  
Not cloning container for unused subsystem ns  
Not cloning container for unused subsystem ns  
Not cloning container for unused subsystem ns

-----[ cut here ]-----

kernel BUG at kernel/workqueue.c:258!

Oops: Exception in kernel mode, sig: 5 [#11]

SMP NR\_CPUS=32 NUMA pSeries

Modules linked in:

NIP: c00000000062e6c LR: c00000000062e54 CTR: 0000000000000000

REGS: c00000000d157af0 TRAP: 0700 Tainted: G D (2.6.22-rc6-mm1)

MSR: 800000000029032 <EE,ME,IR,DR> CR: 24000084 XER: 2000000f

TASK = c00000000d0f7460[33] 'events/6' THREAD: c00000000d154000 CPU: 6

GPR00: 0000000000000001 c00000000d157d70 c00000000060fed8 0000000000000001

GPR04: c0000000005c2938 0000000000000000 c00000000064438 c00000000d005600

GPR08: c000000000684f20 7fffffff000000003506570 c0000000f204fe88

GPR12: 0000000000004000 c000000000529980 0000000000000000 c00000000045e928

GPR16: 4000000001c00000 c00000000045d340 0000000000000000 0000000000000000

GPR20: c00000000050e1d8 000000000210e1d8 000000000210e448 c00000000050e448

GPR24: 0000000001a1fb80 c00000000050c2d8 c00000000d154000 c0000000005c2968

GPR28: c0000000f204fe80 c0000000f2665008 c00000000540eb8 c0000000f2665000

NIP [c00000000062e6c] .run\_workqueue+0xe0/0x208

LR [c00000000062e54] .run\_workqueue+0xc8/0x208

Call Trace:

[c00000000d157d70] [c00000000062ea4] .run\_workqueue+0x118/0x208 (unreliable)

[c00000000d157e10] [c000000000639bc] .worker\_thread+0x114/0x138

[c00000000d157f00] [c00000000068ac8] .kthread+0x78/0xc4

[c00000000d157f90] [c000000000231b4] .kernel\_thread+0x4c/0x68

Instruction dump:

980d01cc 7c2004ac 38000000 38600001 901c0000 4bfa80f9 60000000 e81dfff8

78000764 7c00e278 3120ffff 7c090110 <0b000000> 38000001 7d20f8a8 7d290078

Not cloning container for unused subsystem ns

Not cloning container for unused subsystem ns

Unable to handle kernel paging request for data at address 0x00000008

Faulting instruction address: 0xc000000000d9990

Oops: Kernel access of bad area, sig: 11 [#12]

SMP NR\_CPUS=32 NUMA pSeries

Modules linked in:

NIP: c000000000d9990 LR: c000000000d9a28 CTR: c00000000048060

REGS: c00000000de9b870 TRAP: 0300 Tainted: G D (2.6.22-rc6-mm1)

MSR: 800000000009032 <EE,ME,IR,DR> CR: 24000428 XER: 2000000f

DAR: 0000000000000008, DSISR: 0000000042000000

TASK = c00000000d15cfa0[4871] 'pidns\_exec' THREAD: c00000000de98000 CPU: 2

GPR00: c000000000d9a28 c0000000de9baf0 c0000000060fed8 0000000000000000  
GPR04: c0000000065d708 c0000000100b3f00 c000000000d8eac c00000000631980  
GPR08: c0000000100b3f00 0000000000000000 c00000000966158 c0000000de9bc00  
GPR12: 0000000000000001 c00000000529180 00000000ffa0d094 0000000000000000  
GPR16: 00000000100a0000 00000000100a92c8 00000000100a0000 0000000010080000  
GPR20: 0000000000000000 0000000000000000 0000000000000000 00000000100a9568  
GPR24: 00000000f7fdb60 c0000000065e2c8 c0000000de9bc00 c00000000634500  
GPR28: 0000000000000000 c0000000d27a200 c0000000055ec28 c0000000d27a200  
NIP [c000000000d9990] .release\_mounts+0x3c/0x108  
LR [c000000000d9a28] .release\_mounts+0xd4/0x108  
Call Trace:  
[c0000000de9baf0] [c000000000d9a28] .release\_mounts+0xd4/0x108 (unreliable)  
[c0000000de9bb90] [c000000000d9b38] .\_\_put\_mnt\_ns+0xdc/0x114  
[c0000000de9bc50] [c0000000006cd74] .free\_nsproxy+0x54/0xe8  
[c0000000de9bce0] [c0000000005272c] .do\_exit+0x948/0xa08  
[c0000000de9bda0] [c000000000528c0] .sys\_exit\_group+0x0/0x8  
[c0000000de9be30] [c0000000000872c] syscall\_exit+0x0/0x40  
Instruction dump:  
fb61ffd8 fb81ffe0 fba1ffe8 fbe1fff8 f8010010 f821ff61 ebc2c998 7c7a1b78  
480000a8 e97f0008 e93f0000 f92b0000 <f9690008> fbff0000 fbff0008 e81f0010  
Not cloning container for unused subsystem ns  
Not cloning container for unused subsystem ns  
Not cloning container for unused subsystem ns  
Not cloning container for unused subsystem ns  
Unable to handle kernel paging request for data at address 0x00000020  
Faulting instruction address: 0xc00000000051e88  
Oops: Kernel access of bad area, sig: 11 [#13]  
SMP NR\_CPUS=32 NUMA pSeries  
Modules linked in:  
NIP: c00000000051e88 LR: c00000000051e1c CTR: 80000000013f270  
REGS: c0000000de9b3a0 TRAP: 0300 Tainted: G D (2.6.22-rc6-mm1)  
MSR: 800000000009032 <EE,ME,IR,DR> CR: 28000444 XER: 00000002  
DAR: 0000000000000020, DSISR: 0000000400000000  
TASK = c0000000d15cfa0[4871] 'pidns\_exec' THREAD: c0000000de98000 CPU: 2  
GPR00: 0000000000000000 c0000000de9b620 c0000000060fed8 0000000000000000  
GPR04: 0000000000000000 c0000000d15cfa0 ffffffff 0000000000000000  
GPR08: 0000000000000000 0000000000000000 c0000000063e580 c0000000063e580  
GPR12: 0000000000004000 c00000000529180 00000000ffa0d094 0000000000000000  
GPR16: 00000000100a0000 00000000100a92c8 00000000100a0000 0000000010080000  
GPR20: 0000000000000000 0000000000000000 0000000000000000 00000000100a9568  
GPR24: 00000000f7fdb60 c0000000065e2c8 0000000000000000b 0000000000000000b  
GPR28: c0000000045fad8 c0000000d15cfa0 c00000000540998 0000000000000001  
NIP [c00000000051e88] .do\_exit+0xa4/0xa08  
LR [c00000000051e1c] .do\_exit+0x38/0xa08  
Call Trace:  
[c0000000de9b620] [c00000000051e1c] .do\_exit+0x38/0xa08 (unreliable)  
[c0000000de9b6e0] [c00000000021478] .die+0x20c/0x210  
[c0000000de9b780] [c0000000002971c] .bad\_page\_fault+0xb8/0xd4

```
[c0000000de9b800] [c00000000004e98] handle_page_fault+0x3c/0x58
--- Exception: 300 at .release_mounts+0x3c/0x108
  LR = .release_mounts+0xd4/0x108
[c0000000de9bb90] [c000000000d9b38] .__put_mnt_ns+0xdc/0x114
[c0000000de9bc50] [c00000000006cd74] .free_nsproxy+0x54/0xe8
[c0000000de9bce0] [c00000000005272c] .do_exit+0x948/0xa08
[c0000000de9bda0] [c0000000000528c0] .sys_exit_group+0x0/0x8
[c0000000de9be30] [c00000000000872c] syscall_exit+0x0/0x40
Instruction dump:
801d0198 2f800000 40be0010 e87e8050 4bffb5c9 60000000 e80d01a0 7fa00278
3120ffff 7c090110 0b000000 e93d04a8 <e9290020> e8090828 7fbd0000 40be0048
Not cloning container for unused subsystem ns
Not cloning container for unused subsystem ns
Not cloning container for unused subsystem ns
Not cloning container for unused subsystem ns
Not cloning container for unused subsystem ns
Not cloning container for unused subsystem ns
Unable to handle kernel paging request for data at address 0x00000020
Faulting instruction address: 0xc000000000051e88
Oops: Kernel access of bad area, sig: 11 [#14]
SMP NR_CPUS=32 NUMA pSeries
Modules linked in:
NIP: c000000000051e88 LR: c000000000051e1c CTR: 800000000013f270
REGS: c0000000de9aed0 TRAP: 0300 Tainted: G   D (2.6.22-rc6-mm1)
MSR: 8000000000009032 <EE,ME,IR,DR> CR: 28000444 XER: 00000002
DAR: 0000000000000020, DSISR: 0000000040000000
TASK = c0000000d15cfa0[4871] 'pidns_exec' THREAD: c0000000de98000 CPU: 2
GPR00: 0000000000000000 c0000000de9b150 c00000000060fed8 0000000000000000
GPR04: 0000000000000000 c0000000d15cfa0 ffffffff 0000000000000000
GPR08: 0000000000000000 0000000000000000 c00000000063e580 c00000000063e580
GPR12: 0000000000004000 c000000000529180 00000000ffa0d094 0000000000000000
GPR16: 00000000100a0000 00000000100a92c8 00000000100a0000 0000000010080000
GPR20: 0000000000000000 0000000000000000 0000000000000000 00000000100a9568
GPR24: 00000000f7fdb60 c00000000065e2c8 000000000000000b 000000000000000b
GPR28: c0000000045fad8 c0000000d15cfa0 c000000000540998 0000000000000001
NIP [c000000000051e88] .do_exit+0xa4/0xa08
LR [c000000000051e1c] .do_exit+0x38/0xa08
Call Trace:
[c0000000de9b150] [c000000000051e1c] .do_exit+0x38/0xa08 (unreliable)
[c0000000de9b210] [c000000000021478] .die+0x20c/0x210
[c0000000de9b2b0] [c00000000002971c] .bad_page_fault+0xb8/0xd4
[c0000000de9b330] [c00000000004e98] handle_page_fault+0x3c/0x58
--- Exception: 300 at .do_exit+0xa4/0xa08
  LR = .do_exit+0x38/0xa08
[c0000000de9b6e0] [c000000000021478] .die+0x20c/0x210
[c0000000de9b780] [c00000000002971c] .bad_page_fault+0xb8/0xd4
[c0000000de9b800] [c00000000004e98] handle_page_fault+0x3c/0x58
--- Exception: 300 at .release_mounts+0x3c/0x108
```



```
LR = .release_mounts+0xd4/0x108
[c0000000de9bb90] [c000000000d9b38] .__put_mnt_ns+0xdc/0x114
[c0000000de9bc50] [c00000000006cd74] .free_nsproxy+0x54/0xe8
[c0000000de9bce0] [c00000000005272c] .do_exit+0x948/0xa08
[c0000000de9bda0] [c0000000000528c0] .sys_exit_group+0x0/0x8
[c0000000de9be30] [c00000000000872c] syscall_exit+0x0/0x40
Instruction dump:
801d0198 2f800000 40be0010 e87e8050 4bffb5c9 60000000 e80d01a0 7fa00278
3120ffff 7c090110 0b000000 e93d04a8 <e9290020> e8090828 7fbd0000 40be0048
Not cloning container for unused subsystem ns
Not cloning container for unused subsystem ns
Not cloning container for unused subsystem ns
Not cloning container for unused subsystem ns
Not cloning container for unused subsystem ns
Not cloning container for unused subsystem ns
Unable to handle kernel paging request for data at address 0x00000020
Faulting instruction address: 0xc000000000051e88
Oops: Kernel access of bad area, sig: 11 [#15]
SMP NR_CPUS=32 NUMA pSeries
Modules linked in:
NIP: c000000000051e88 LR: c000000000051e1c CTR: 800000000013f270
REGS: c0000000de9aa00 TRAP: 0300 Tainted: G D (2.6.22-rc6-mm1)
MSR: 8000000000009032 <EE,ME,IR,DR> CR: 28000444 XER: 00000002
DAR: 0000000000000020, DSISR: 0000000040000000
TASK = c0000000d15cfa0[4871] 'pidns_exec' THREAD: c0000000de98000 CPU: 2
GPR00: 0000000000000000 c0000000de9ac80 c0000000060fed8 0000000000000000
GPR04: 0000000000000000 c0000000d15cfa0 ffffffff0000000000000000
GPR08: 0000000000000000 0000000000000000 c0000000063e580 c0000000063e580
GPR12: 0000000000004000 c00000000529180 0000000ffa0d094 0000000000000000
GPR16: 00000000100a0000 00000000100a92c8 00000000100a0000 0000000010080000
GPR20: 0000000000000000 0000000000000000 0000000000000000 00000000100a9568
GPR24: 00000000f7fdbb60 c0000000065e2c8 000000000000000b 000000000000000b
GPR28: c0000000045fad8 c0000000d15cfa0 c00000000540998 0000000000000001
NIP [c000000000051e88] .do_exit+0xa4/0xa08
LR [c000000000051e1c] .do_exit+0x38/0xa08
Call Trace:
[c0000000de9ac80] [c000000000051e1c] .do_exit+0x38/0xa08 (unreliable)
[c0000000de9ad40] [c000000000021478] .die+0x20c/0x210
[c0000000de9ade0] [c00000000002971c] .bad_page_fault+0xb8/0xd4
[c0000000de9ae60] [c00000000004e98] handle_page_fault+0x3c/0x58
--- Exception: 300 at .do_exit+0xa4/0xa08
LR = .do_exit+0x38/0xa08
[c0000000de9b210] [c000000000021478] .die+0x20c/0x210
[c0000000de9b2b0] [c00000000002971c] .bad_page_fault+0xb8/0xd4
[c0000000de9b330] [c00000000004e98] handle_page_fault+0x3c/0x58
--- Exception: 300 at .do_exit+0xa4/0xa08
LR = .do_exit+0x38/0xa08
[c0000000de9b6e0] [c000000000021478] .die+0x20c/0x210
```

```
[c0000000de9b780] [c0000000002971c] .bad_page_fault+0xb8/0xd4
[c0000000de9b800] [c00000000004e98] handle_page_fault+0x3c/0x58
--- Exception: 300 at .release_mounts+0x3c/0x108
  LR = .release_mounts+0xd4/0x108
[c0000000de9bb90] [c000000000d9b38] .__put_mnt_ns+0xdc/0x114
[c0000000de9bc50] [c00000000006cd74] .free_nsproxy+0x54/0xe8
[c0000000de9bce0] [c00000000005272c] .do_exit+0x948/0xa08
[c0000000de9bda0] [c0000000000528c0] .sys_exit_group+0x0/0x8
[c0000000de9be30] [c0000000000872c] syscall_exit+0x0/0x40
Instruction dump:
801d0198 2f800000 40be0010 e87e8050 4bffb5c9 60000000 e80d01a0 7fa00278
3120ffff 7c090110 0b000000 e93d04a8 <e9290020> e8090828 7fbd0000 40be0048
Not cloning container for unused subsystem ns
Not cloning container for unused subsystem ns
Not cloning container for unused subsystem ns
Not cloning container for unused subsystem ns
Not cloning container for unused subsystem ns
Not cloning container for unused subsystem ns
Unable to handle kernel paging request for data at address 0x00000020
Faulting instruction address: 0xc000000000051e88
Oops: Kernel access of bad area, sig: 11 [#16]
SMP NR_CPUS=32 NUMA pSeries
Modules linked in:
NIP: c000000000051e88 LR: c000000000051e1c CTR: 800000000013f270
REGS: c0000000de9a530 TRAP: 0300 Tainted: G   D (2.6.22-rc6-mm1)
MSR: 8000000000009032 <EE,ME,IR,DR> CR: 28000444 XER: 00000002
DAR: 0000000000000020, DSISR: 0000000040000000
TASK = c0000000d15cfa0[4871] 'pidns_exec' THREAD: c0000000de98000 CPU: 2
GPR00: 0000000000000000 c0000000de9a7b0 c00000000060fed8 0000000000000000
GPR04: 0000000000000000 c0000000d15cfa0 ffffffff0000000000000000
GPR08: 0000000000000000 0000000000000000 c00000000063e580 c00000000063e580
GPR12: 0000000000004000 c000000000529180 00000000ffa0d094 0000000000000000
GPR16: 00000000100a0000 00000000100a92c8 00000000100a0000 0000000010080000
GPR20: 0000000000000000 0000000000000000 0000000000000000 00000000100a9568
GPR24: 00000000f7fdb60 c00000000065e2c8 0000000000000000b 0000000000000000b
GPR28: c0000000045fad8 c0000000d15cfa0 c000000000540998 0000000000000001
NIP [c000000000051e88] .do_exit+0xa4/0xa08
LR [c000000000051e1c] .do_exit+0x38/0xa08
Call Trace:
[c0000000de9a7b0] [c000000000051e1c] .do_exit+0x38/0xa08 (unreliable)
[c0000000de9a870] [c000000000021478] .die+0x20c/0x210
[c0000000de9a910] [c00000000002971c] .bad_page_fault+0xb8/0xd4
[c0000000de9a990] [c00000000004e98] handle_page_fault+0x3c/0x58
--- Exception: 300 at .do_exit+0xa4/0xa08
  LR = .do_exit+0x38/0xa08
[c0000000de9ad40] [c000000000021478] .die+0x20c/0x210
[c0000000de9ade0] [c00000000002971c] .bad_page_fault+0xb8/0xd4
[c0000000de9ae60] [c00000000004e98] handle_page_fault+0x3c/0x58
```

```
--- Exception: 300 at .do_exit+0xa4/0xa08
  LR = .do_exit+0x38/0xa08
[c0000000de9b210] [c00000000021478] .die+0x20c/0x210
[c0000000de9b2b0] [c0000000002971c] .bad_page_fault+0xb8/0xd4
[c0000000de9b330] [c00000000004e98] handle_page_fault+0x3c/0x58
--- Exception: 300 at .do_exit+0xa4/0xa08
  LR = .do_exit+0x38/0xa08
[c0000000de9b6e0] [c00000000021478] .die+0x20c/0x210
[c0000000de9b780] [c0000000002971c] .bad_page_fault+0xb8/0xd4
[c0000000de9b800] [c00000000004e98] handle_page_fault+0x3c/0x58
--- Exception: 300 at .release_mounts+0x3c/0x108
  LR = .release_mounts+0xd4/0x108
[c0000000de9bb90] [c000000000d9b38] .__put_mnt_ns+0xdc/0x114
[c0000000de9bc50] [c0000000006cd74] .free_nsproxy+0x54/0xe8
[c0000000de9bce0] [c0000000005272c] .do_exit+0x948/0xa08
[c0000000de9bda0] [c000000000528c0] .sys_exit_group+0x0/0x8
[c0000000de9be30] [c0000000000872c] syscall_exit+0x0/0x40
Instruction dump:
801d0198 2f800000 40be0010 e87e8050 4bffb5c9 60000000 e80d01a0 7fa00278
3120ffff 7c090110 0b000000 e93d04a8 <e9290020> e8090828 7fbd0000 40be0048
...
```

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Pavel Emelianov](#) on Mon, 16 Jul 2007 08:47:10 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

> My x86\_64 system boots fine but crashes as below, when I run my  
> 'pidns\_exec' test with a simple program that prints getpid(), getppid()  
> etc of the process in the child pid ns.  
>  
> Pls see  
>  
> <http://www.geocities.com/sukadevb/Pidspace/2.6.22-rc6-mm1-pavel-1.tgz>  
>  
> for the patches I currently have applied and let me know if I need more  
> on top.  
>  
> And see  
>  
> <http://www.geocities.com/sukadevb/Pidspace/Test1/>  
>

```

> for the test programs. You may need to run the 'mypid-loop.x' script
> to repro the crash. The pidns_exec.c program calls clone() with CLONE_NEWPID
> and execs the given program (it was included in Patch 0 of the patchset I
> posted to Containers).
>
> Suka
>
> login: Unable to handle kernel NULL pointer dereference at 00000000000002fc RIP:
> [<ffffffff802b9e5e>] proc_get_sb+0xfb/0x138
> PGD 104d53067 PUD 104d4d067 PMD 0
> Oops: 0002 [1] SMP
> CPU 2
> Modules linked in:
> Pid: 3279, comm: pidns_exec Not tainted 2.6.22-rc6-mm1-ovz1 #10
> RIP: 0010:[<ffffffff802b9e5e>] [<ffffffff802b9e5e>] proc_get_sb+0xfb/0x138
> RSP: 0018:ffff8101029d7d28 EFLAGS: 00010202
> RAX: ffff810100651840 RBX: ffff810104461400 RCX: ffff810100651878
> RDX: 0000000000000000 RSI: ffffffff806e5460 RDI: 0000000000000238
> RBP: ffff810102d886f8 R08: ffff810104461400 R09: ffff810100026000
> R10: 0000000000000000 R11: 0000000000000002 R12: ffff8101029c6000
> R13: 0000000002000000 R14: ffffffff806ee920 R15: ffff810102088cc0
> FS: 00002b0b499ec6f0(0000) GS:ffff81010069c3c0(0000) knlGS:0000000000000000
> CS: 0010 DS: 0000 ES: 0000 CR0: 000000008005003b
> CR2: 00000000000002fc CR3: 000000010381b000 CR4: 000000000000006e0
> DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
> DR3: 0000000000000000 DR6: 00000000ffff0ff0 DR7: 0000000000000400
> Process pidns_exec (pid: 3279, threadinfo ffff8101029d6000, task ffff81010269e7f0)
> Stack: ffff810102088cc0 ffff810102088cc0 00000000ffffff4 ffffffff806ee920
> ffffffff8065f9d9 ffff8101029c6000 0000000002000000 ffffffff80287164
> 00000000000000d0 ffff8101029c6000 ffffffff806e5460 ffff8101029c6000
> Call Trace:
> [<ffffffff80287164>] vfs_kern_mount+0x4f/0x8b
> [<ffffffff802b9cf4>] pid_ns_prepare_proc+0x13/0x2e
> [<ffffffff80245be3>] copy_pid_ns+0xd7/0x164
> [<ffffffff8024af34>] create_new_namespaces+0xde/0x192
> [<ffffffff8024b0aa>] copy_namespaces+0x4b/0x85
> [<ffffffff802347e2>] copy_process+0xcb4/0x1439
> [<ffffffff8020bbee>] system_call+0x7e/0x83
> [<ffffffff8023556a>] do_fork+0x6c/0x1e7
> [<ffffffff8020bf07>] ptregscall_common+0x67/0xb0
>

```

Here's the patch fixing the problem. So, Suka, I propose that you review my patches, point out things that you don't like and would like to see your code instead. After all I will re-split the set with your fixes, mark some patches with your From: and send them to Andrew. What do you think?

---

```
--- ./fs/proc/root.c.procpidnsfix 2007-07-16 10:32:00.000000000 +0400
+++ ./fs/proc/root.c 2007-07-16 12:34:35.000000000 +0400
@@ -79,8 +79,6 @@ static int proc_get_sb(struct file_syste
    if (!ei->pid)
        ei->pid = find_get_pid(1);
    sb->s_flags |= MS_ACTIVE;
-
- mntput(ns->proc_mnt);
  ns->proc_mnt = mnt;
}
```

---

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

---

---

Subject: Re: [PATCH 0/16] Pid namespaces  
Posted by [Sukadev Bhattiprolu](#) on Tue, 17 Jul 2007 04:23:39 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Pavel Emelianov [xemul@openvz.org] wrote:

```
| >My x86_64 system boots fine but crashes as below, when I run my
| >'pidns_exec' test with a simple program that prints getpid(), getppid()
| >etc of the process in the child pid ns.
| >
| >Pls see
| >
| >http://www.geocities.com/sukadevb/Pidspace/2.6.22-rc6-mm1-pavel-1.tgz
| >
| >for the patches I currently have applied and let me know if I need more
| >on top.
| >
| >And see
| >
| >http://www.geocities.com/sukadevb/Pidspace/Test1/
| >
| >for the test programs. You may need to run the 'mypid-loop.x' script
| >to repro the crash. The pidns_exec.c program calls clone() with
| >CLONE_NEWPID
| >and execs the given program (it was included in Patch 0 of the patchset I
| >posted to Containers).
| >
| >Suka
| >
| >login: Unable to handle kernel NULL pointer dereference at
| >00000000000002fc RIP:
```

```

| > [<ffffffff802b9e5e>] proc_get_sb+0xfb/0x138
| >PGD 104d53067 PUD 104d4d067 PMD 0
| >Oops: 0002 [1] SMP
| >CPU 2
| >Modules linked in:
| >Pid: 3279, comm: pidns_exec Not tainted 2.6.22-rc6-mm1-ovz1 #10
| >RIP: 0010:[<ffffffff802b9e5e>] [<ffffffff802b9e5e>] proc_get_sb+0xfb/0x138
| >RSP: 0018:ffff8101029d7d28 EFLAGS: 00010202
| >RAX: ffff810100651840 RBX: ffff810104461400 RCX: ffff810100651878
| >RDX: 0000000000000000 RSI: ffffffff806e5460 RDI: 0000000000000238
| >RBP: ffff810102d886f8 R08: ffff810104461400 R09: ffff810100026000
| >R10: 0000000000000000 R11: 0000000000000002 R12: ffff8101029c6000
| >R13: 0000000002000000 R14: ffffffff806ee920 R15: ffff810102088cc0
| >FS: 00002b0b499ec6f0(0000) GS:ffff81010069c3c0(0000)
| >knlGS:0000000000000000
| >CS: 0010 DS: 0000 ES: 0000 CR0: 000000008005003b
| >CR2: 00000000000002fc CR3: 000000010381b000 CR4: 00000000000006e0
| >DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
| >DR3: 0000000000000000 DR6: 00000000ffff0ff0 DR7: 0000000000000400
| >Process pidns_exec (pid: 3279, threadinfo ffff8101029d6000, task
| >ffff81010269e7f0)
| >Stack: ffff810102088cc0 ffff810102088cc0 00000000ffffff4 ffffffff806ee920
| > ffffffff8065f9d9 ffff8101029c6000 0000000002000000 ffffffff80287164
| > 00000000000000d0 ffff8101029c6000 ffffffff806e5460 ffff8101029c6000
| >Call Trace:
| > [<ffffffff80287164>] vfs_kern_mount+0x4f/0x8b
| > [<ffffffff802b9cf4>] pid_ns_prepare_proc+0x13/0x2e
| > [<ffffffff80245be3>] copy_pid_ns+0xd7/0x164
| > [<ffffffff8024af34>] create_new_namespaces+0xde/0x192
| > [<ffffffff8024b0aa>] copy_namespaces+0x4b/0x85
| > [<ffffffff802347e2>] copy_process+0xcb4/0x1439
| > [<ffffffff8020bbee>] system_call+0x7e/0x83
| > [<ffffffff8023556a>] do_fork+0x6c/0x1e7
| > [<ffffffff8020bf07>] ptregscall_common+0x67/0xb0
| >
|

```

| Here's the patch fixing the problem.

Yes, I think it fixes the problem I was seeing before.  
However, my next test failed. This time I run:

```

$ cat ./lxc-wrap.sh
#!/bin/sh

```

```

if [ $$ -eq 1 ]; then
  echo "$$ ( `basename $0` ): Executing in nested pid ns..."
fi

```

```
port_num=$1;
if [ $# -lt 1 ]; then
    port_num=2709
fi
```

```
mount -t proc lxcproc /proc
/usr/sbin/sshd -D -p $port_num
umount /proc
```

```
$ ./pidns_exec ./lxc-wrap.sh
```

This creates a new pid ns and an sshd in that ns. From another window I ssh to the system with port number 2709 and I am in the new pid ns. (a simple prog to print getpid(), getppid() etc seems to be fine).

But "ps -e" fails:

```
$ ps -e
Error: /proc must be mounted
To mount /proc at boot you need an /etc/fstab line like:
    /proc /proc proc defaults
In the meantime, mount /proc /proc -t proc
```

```
$ mount -t proc none /proc
mount: /proc is busy
```

If I comment out the "mount" command in the lxc-wrap.sh script above, this mount seems to work, but "ps -e" still reports the same error.

| So, Suka, I propose that you review my patches, point out things that you  
| don't like and would like to see your code instead.

I am reviewing them and I thought we wanted to start with my patchset. I started porting my patches to rc6-mm1 plus patches we already sent to -mm. Well, I can hold off on that and review/test your patches.

| After all I will re-split the set with your fixes, mark  
| some patches with your From: and send them to Andrew. What do you think?

I don't mind re-splitting the patches, but would that be more work for us and would Andrew and other reviewers prefer patches split logically. For instance if you and I contributed to a patch, can we just put both our Signed-off on one patch rather than splitting it ?

```
|
| ---
|
```

```
| --- ./fs/proc/root.c.procpidnsfix 2007-07-16 10:32:00.000000000 +0400
| +++ ./fs/proc/root.c 2007-07-16 12:34:35.000000000 +0400
| @@ -79,8 +79,6 @@ static int proc_get_sb(struct file_syste
| if (!ei->pid)
|   ei->pid = find_get_pid(1);
|   sb->s_flags |= MS_ACTIVE;
| -
| - mntput(ns->proc_mnt);
|   ns->proc_mnt = mnt;
| }
```

---

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

---

---

Subject: Re: [PATCH 13/16] Switch to operating with pid\_numbers instead of pids  
Posted by [Sukadev Bhattiprolu](#) on Wed, 25 Jul 2007 00:36:19 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Pavel Emelianov [xemul@openvz.org] wrote:  
| 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_namespa
| {
|   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;

```

If pid\_ns->level is say 3 and alloc\_pidmap() succeeds when i=0,1 and fails when i=2, we would try to free\_pidmap() even from pid->pid\_number[2].pid\_ns. This would incorrectly a) drop reference count on that pid namespace, and incorrectly increment pidmap->nr\_free.

Should we use kmem\_cache\_zalloc() and check for a non-NULL pid\_ns before calling free\_pidmap() below ?

```
|
| + 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);
```

i.e all pid->numbers[] may not be initialized here right ?

---

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

---

---

Subject: Re: [PATCH 13/16] Switch to operating with pid\_numbers instead of pids  
Posted by [Pavel Emelianov](#) on Wed, 25 Jul 2007 10:07:04 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

sukadev@us.ibm.com wrote:

> Pavel Emelianov [xemul@openvz.org] wrote:

> | 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_namespa
> | {
> | 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_cache, GFP_KERNEL);
> | + pid = kmem_cache_alloc(pid_ns->pid_cache, 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;
> |
> |
> | If pid_ns->level is say 3 and alloc_pidmap() succeeds when i=0,1

```

It cannot :) If level is 3, then we'll allocate for 3, 2, 1, 0 sequence.  
The loop is descending, not ascending...

```

> and fails when i=2, we would try to free_pidmap() even from
> pid->pid_number[2].pid_ns. This would incorrectly a)
> drop reference count on that pid namespace, and incorrectly
> increment pidmap->nr_free.
>
> Should we use kmem_cache_zalloc() and check for a non-NULL pid_ns
> before calling free_pidmap() below ?
>
> |
> | + 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);
>
> i.e all pid->numbers[] may not be initialized here right ?

```

The numbers from i up to pid->level are initialized, so this loop looks correct.

Thanks,  
Pavel

---

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

---

Subject: Re: [PATCH 13/16] Switch to operating with pid\_numbers instead of pids  
Posted by [Sukadev Bhattiprolu](#) on Wed, 25 Jul 2007 19:13:34 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Pavel Emelianov [xemul@openvz.org] wrote:  
| sukadev@us.ibm.com wrote:  
| > Pavel Emelianov [xemul@openvz.org] wrote:  
| > | 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_cache, 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_namespa
|> {
|> 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;
|>
|>
|> If pid_ns->level is say 3 and alloc_pidmap() succeeds when i=0,1
|
| It cannot :) If level is 3, then we'll allocate for 3, 2, 1, 0 sequence.
| The loop is descending, not ascending...

```

Aah descending - thats right. But I still think there is a problem.

Here is your code that I am referring to:

```

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

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;

```

<snip>

```
out:
return pid;
out_free:
    for (i++; i <= pid->level; i++)
        free_pidmap(pid->numbers[i].ns, pid->numbers[i].nr);

    kmem_cache_free(pid_ns->pid_cache, pid);
    pid = NULL;
goto out;
```

<end code>

Lets say initially pid\_ns->level = 3 and alloc\_pidmap() succeeds for i=3 and i=2 but fails for i=1 and we execute "goto out\_free".

But pid->level is uninitialized at this point right ?

Even if it were set to zero (using kmem\_cache\_zalloc()), we may not free the two pidmap entries we allocated for i=3 and i=2.

Suka

---

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

---

---

Subject: Re: [PATCH 13/16] Switch to operating with pid\_numbers instead of pids  
Posted by [Pavel Emelianov](#) on Thu, 26 Jul 2007 06:42:22 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

sukadev@us.ibm.com wrote:  
> Pavel Emelianov [xemul@openvz.org] wrote:  
> | sukadev@us.ibm.com wrote:  
> | > Pavel Emelianov [xemul@openvz.org] wrote:  
> | >| 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_namespa
> |>| {
> |>|     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;
> |>|
> |>|
> |>|> If pid_ns->level is say 3 and alloc_pidmap() succeeds when i=0,1
> |>|
> |>| It cannot :) If level is 3, then we'll allocate for 3, 2, 1, 0 sequence.
> |>| The loop is descending, not ascending...
>
> Aah descending - thats right. But I still think there is a problem.
>
> Here is your code that I am referring to:
>
>     pid = kmem_cache_alloc(pid_ns->pid_cachep, GFP_KERNEL);
>     if (!pid)
>         goto out;
>
>     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;
>
> <snip>
>
> out:
> return pid;
> out_free:
>     for (i++; i <= pid->level; i++)

```

```
>         free_pidmap(pid->numbers[i].ns, pid->numbers[i].nr);
>
>     kmem_cache_free(pid_ns->pid_cache, pid);
>     pid = NULL;
>     goto out;
>
> <end code>
>
> Lets say initially pid_ns->level = 3 and alloc_pidmap() succeeds for
> i=3 and i=2 but fails for i=1 and we execute "goto out_free".
>
> But pid->level is uninitialized at this point right ?
>
> Even if it were set to zero (using kmem_cache_zalloc()), we may not
> free the two pidmap entries we allocated for i=3 and i=2.
```

Yes. I found this after detailed look at the code and (hope) fixed.

```
> Suka
>
```

Thanks,  
Pavel

---

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

---