Subject: [PATCH 19/20] Changes to show virtual ids to user
Posted by Pavel Emelianov on Fri, 10 Aug 2007 11:48:28 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 Emelyanov <xemul@openvz.org>
Cc: Oleg Nesterov <oleg@tv-sign.ru>

```
 arch/ia64/kernel/signal.c        |    4 +-
 arch/parisc/kernel/signal.c      |    2 -
 arch/sparc/kernel/sys_sunos.c    |    2 -
 arch/sparc64/kernel/sys_sunos32.c |    2 -
 drivers/char/tty_io.c            |    7 ++---
 fs/binfmt_elf.c                  |   16 +++++------
 fs/binfmt_elf_fdpic.c            |   16 +++++------
 fs/exec.c                        |    4 +-
 fs/fcntl.c                       |    5 ++-
 fs/ioprio.c                      |    5 ++-
 fs/proc/array.c                  |   27 +++++++++++++------
 fs/proc/base.c                   |   37 ++++++++++++++++-----------
 include/net/scm.h                |    4 ++
 ipc/mqueue.c                     |    7 +++--
 ipc/msg.c                        |    6 ++--
 ipc/sem.c                        |    8 ++---
 ipc/shm.c                        |    6 ++--
 kernel/capability.c              |   14 +++++-----
 kernel/exit.c                    |   31 ++++++++++++++--------
 kernel/futex.c                   |   20 ++++++++------
 kernel/ptrace.c                  |    4 ++
 kernel/sched.c                   |    3 +-
 kernel/signal.c                  |   52 ++++++++++++++++++++++++++++----------
 kernel/sys.c                     |   42 ++++++++++++++++++------------
 kernel/sysctl.c                  |    2 -
```

```
 kernel/timer.c                  |    7 ++---
 net/core/scm.c                  |    4 ++
 net/unix/af_unix.c              |    6 ++--
 28 files changed, 212 insertions(+), 131 deletions(-)

--- ./arch/ia64/kernel/signal.c.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./arch/ia64/kernel/signal.c 2007-08-10 12:43:21.000000000 +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.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./arch/parisc/kernel/signal.c 2007-08-10 12:43:21.000000000 +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);
--- ./arch/sparc/kernel/sys_sunos.c.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./arch/sparc/kernel/sys_sunos.c 2007-08-10 12:43:21.000000000 +0400
@@ -866,7 +866,7 @@ asmlinkage int sunos_killpg(int pgrp, in
  rcu_read_lock();
  ret = -EINVAL;
  if (pgrp > 0)
-  ret = kill_pgrp(find_pid(pgrp), sig, 0);
+  ret = kill_pgrp(find_vpid(pgrp), sig, 0);
  rcu_read_unlock();

  return ret;
--- ./arch/sparc64/kernel/sys_sunos32.c.ve18 2007-08-10 12:40:59.000000000 +0400
```

```
+++ ./arch/sparc64/kernel/sys_sunos32.c 2007-08-10 12:43:21.000000000 +0400
@@ -831,7 +831,7 @@ asmlinkage int sunos_killpg(int pgrp, in
  rcu_read_lock();
  ret = -EINVAL;
  if (pgrp > 0)
-  ret = kill_pgrp(find_pid(pgrp), sig, 0);
+  ret = kill_pgrp(find_vpid(pgrp), sig, 0);
  rcu_read_unlock();

  return ret;
--- ./drivers/char/tty_io.c.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./drivers/char/tty_io.c 2007-08-10 12:43:21.000000000 +0400
@@ -103,6 +103,7 @@
 #include <linux/selection.h>

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

 #undef TTY_DEBUG_HANGUP

@@ -3109,7 +3110,7 @@ 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);
 }

 /**
@@ -3143,7 +3144,7 @@ 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;
@@ -3180,7 +3181,7 @@ 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.ve18 2007-08-10 12:40:59.000000000 +0400
```

```
+++ ./fs/binfmt_elf.c 2007-08-10 12:43:21.000000000 +0400
@@ -1380,10 +1380,10 @@ 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
@@ -1426,10 +1426,10 @@ 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.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./fs/binfmt_elf_fdpic.c 2007-08-10 12:43:21.000000000 +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.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./fs/exec.c 2007-08-10 12:43:21.000000000 +0400
@@ -1463,7 +1463,7 @@ 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;
@@ -1543,7 +1543,7 @@ static int format_corename(char *corenam
  if (!ispipe && !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/fcntl.c.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./fs/fcntl.c 2007-08-10 12:43:21.000000000 +0400
@@ -18,6 +18,7 @@
 #include <linux/ptrace.h>
 #include <linux/signal.h>
 #include <linux/rcupdate.h>
+#include <linux/pid_namespace.h>

 #include <asm/poll.h>
 #include <asm/siginfo.h>
@@ -289,7 +290,7 @@ int f_setown(struct file *filp, unsigned
   who = -who;
  }
```

```
  rcu_read_lock();
- pid = find_pid(who);
+ pid = find_vpid(who);
  result = __f_setown(filp, pid, type, force);
  rcu_read_unlock();
  return result;
@@ -305,7 +306,7 @@ pid_t f_getown(struct file *filp)
 {
  pid_t pid;
  read_lock(&filp->f_owner.lock);
- pid = pid_nr(filp->f_owner.pid);
+ pid = pid_nr_ns(filp->f_owner.pid, current->nsproxy->pid_ns);
  if (filp->f_owner.pid_type == PIDTYPE_PGID)
   pid = -pid;
  read_unlock(&filp->f_owner.lock);
--- ./fs/ioprio.c.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./fs/ioprio.c 2007-08-10 12:43:21.000000000 +0400
@@ -25,6 +25,7 @@
 #include <linux/capability.h>
 #include <linux/syscalls.h>
 #include <linux/security.h>
+#include <linux/pid_namespace.h>

 static int set_task_ioprio(struct task_struct *task, int ioprio)
 {
@@ -101,7 +102,7 @@ asmlinkage long sys_ioprio_set(int which
    if (!who)
     pgrp = task_pgrp(current);
    else
-    pgrp = find_pid(who);
+    pgrp = find_vpid(who);
    do_each_pid_task(pgrp, PIDTYPE_PGID, p) {
     ret = set_task_ioprio(p, ioprio);
     if (ret)
@@ -188,7 +189,7 @@ asmlinkage long sys_ioprio_get(int which
    if (!who)
     pgrp = task_pgrp(current);
    else
-    pgrp = find_pid(who);
+    pgrp = find_vpid(who);
    do_each_pid_task(pgrp, PIDTYPE_PGID, p) {
     tmpio = get_task_ioprio(p);
     if (tmpio < 0)
--- ./fs/proc/array.c.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./fs/proc/array.c 2007-08-10 12:43:21.000000000 +0400
@@ -77,6 +77,7 @@
 #include <linux/cpuset.h>
 #include <linux/rcupdate.h>
```

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

 #include <asm/pgtable.h>
 #include <asm/processor.h>
@@ -161,8 +162,15 @@ static inline char *task_state(struct ta
  struct group_info *group_info;
  int g;
  struct fdtable *fdt = NULL;
+ struct pid_namespace *ns;
+ pid_t ppid, tpid;

+ ns = current->nsproxy->pid_ns;
  rcu_read_lock();
+ ppid = pid_alive(p) ?
+  task_tgid_nr_ns(rcu_dereference(p->real_parent), ns) : 0;
+ tpid = pid_alive(p) && p->ptrace ?
+  task_ppid_nr_ns(rcu_dereference(p->parent), ns) : 0;
  buffer += sprintf(buffer,
   "State:\t%s\n"
   "Tgid:\t%d\n"
@@ -172,9 +180,9 @@ static inline char *task_state(struct ta
   "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),
+  ppid, tpid,
   p->uid, p->euid, p->suid, p->fsuid,
   p->gid, p->egid, p->sgid, p->fsgid);

@@ -373,6 +381,9 @@ static int do_task_stat(struct task_stru
  unsigned long rsslim = 0;
  char tcomm[sizeof(task->comm)];
  unsigned long flags;
+ struct pid_namespace *ns;
+
+ ns = current->nsproxy->pid_ns;

  state = *get_task_state(task);
  vsize = eip = esp = 0;
@@ -395,7 +406,7 @@ static int do_task_stat(struct task_stru
   struct signal_struct *sig = task->signal;

   if (sig->tty) {
```

```
-   tty_pgrp = pid_nr(sig->tty->pgrp);
+   tty_pgrp = pid_nr_ns(sig->tty->pgrp, ns);
    tty_nr = new_encode_dev(tty_devnum(sig->tty));
   }

@@ -425,9 +436,9 @@ 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);
  }
@@ -458,7 +469,7 @@ 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 %d 0 %llu %lu %ld %lu %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, ns),
   tcomm,
   state,
   ppid,
--- ./fs/proc/base.c.ve18 2007-08-10 12:41:04.000000000 +0400
+++ ./fs/proc/base.c 2007-08-10 12:43:21.000000000 +0400
@@ -1845,14 +1845,14 @@ 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));
 }

@@ -2227,6 +2227,7 @@ 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)
@@ -2236,8 +2237,9 @@ 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();
@@ -2254,7 +2256,8 @@ 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;
@@ -2262,9 +2265,9 @@ 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
@@ -2304,6 +2307,7 @@ 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;
@@ -2314,11 +2318,12 @@ 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);
@@ -2445,6 +2450,7 @@ 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;
@@ -2453,8 +2459,9 @@ 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();
@@ -2485,14 +2492,14 @@ 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;
  }
@@ -2561,6 +2568,7 @@ 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)
@@ -2594,12 +2602,13 @@ 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.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./include/net/scm.h 2007-08-10 12:43:21.000000000 +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.ve18 2007-08-10 12:40:59.000000000 +0400
```

```
+++ ./ipc/mqueue.c 2007-08-10 12:43:21.000000000 +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"
@@ -336,7 +338,8 @@ static ssize_t mqueue_read_file(struct f
     (info->notify_owner &&
      info->notify.sigev_notify == SIGEV_SIGNAL) ?
     info->notify.sigev_signo : 0,
-    pid_nr(info->notify_owner));
+    pid_nr_ns(info->notify_owner,
+     current->nsproxy->pid_ns));
   spin_unlock(&info->lock);
   buffer[sizeof(buffer)-1] = '\0';
   slen = strlen(buffer)+1;
@@ -513,7 +516,7 @@ 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.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./ipc/msg.c 2007-08-10 12:43:21.000000000 +0400
@@ -611,7 +611,7 @@ static inline int pipelined_send(struct
     msr->r_msg = ERR_PTR(-E2BIG);
    } else {
     msr->r_msg = NULL;
-    msq->q_lrpid = msr->r_tsk->pid;
+    msq->q_lrpid = task_pid_vnr(msr->r_tsk);
     msq->q_rtime = get_seconds();
     wake_up_process(msr->r_tsk);
     smp_mb();
@@ -695,7 +695,7 @@ 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)) {
@@ -810,7 +810,7 @@ long do_msgrcv(int msqid, long *pmtype,
    list_del(&msg->m_list);
    msq->q_qnum--;
    msq->q_rtime = get_seconds();
-   msq->q_lrpid = current->tgid;
+   msq->q_lrpid = task_tgid_vnr(current);
    msq->q_cbytes -= msg->m_ts;
    atomic_sub(msg->m_ts, &msg_bytes);
    atomic_dec(&msg_hdrs);
--- ./ipc/sem.c.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./ipc/sem.c 2007-08-10 12:43:21.000000000 +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);
@@ -1196,7 +1196,7 @@ 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);
@@ -1211,7 +1211,7 @@ 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)
@@ -1382,7 +1382,7 @@ 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.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./ipc/shm.c 2007-08-10 12:43:21.000000000 +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 &&
@@ -392,7 +392,7 @@ 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.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./kernel/capability.c 2007-08-10 12:43:21.000000000 +0400
@@ -68,8 +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;
@@ -102,7 +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) {
@@ -191,7 +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)) ||
@@ -202,8 +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.ve18 2007-08-10 12:41:04.000000000 +0400
+++ ./kernel/exit.c 2007-08-10 12:43:39.000000000 +0400
@@ -1112,15 +1112,17 @@ 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;
  }
```

```
@@ -1190,9 +1192,12 @@ static int wait_task_zombie(struct task_
 {
  unsigned long state;
  int retval, status, traced;
+ 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;
@@ -1313,11 +1318,11 @@ 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)
-  retval = p->pid;
+  retval = task_pid_nr_ns(p, ns);

  if (traced) {
   write_lock_irq(&tasklist_lock);
@@ -1354,6 +1359,7 @@ 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;
@@ -1372,11 +1378,12 @@ 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;

@@ -1447,11 +1454,11 @@ 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);
@@ -1471,6 +1478,7 @@ 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;
@@ -1488,7 +1496,8 @@ static int wait_task_continued(struct ta
   p->signal->flags &= ~SIGNAL_STOP_CONTINUED;
  spin_unlock_irq(&p->sighand->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);
@@ -1499,7 +1508,7 @@ 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/futex.c.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./kernel/futex.c 2007-08-10 12:43:21.000000000 +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"
@@ -652,7 +654,7 @@ 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);

@@ -1105,7 +1107,7 @@ 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;
@@ -1367,7 +1369,7 @@ 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);

@@ -1378,7 +1380,7 @@ 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;
  }
@@ -1407,7 +1409,7 @@ 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;
  }
@@ -1586,7 +1588,7 @@ 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:
@@ -1607,7 +1609,7 @@ 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))
@@ -1616,7 +1618,7 @@ 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;

  /*
@@ -1886,7 +1888,7 @@ 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.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./kernel/ptrace.c 2007-08-10 12:43:21.000000000 +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>
@@ -442,7 +443,8 @@ 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.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./kernel/sched.c 2007-08-10 12:43:21.000000000 +0400
@@ -63,6 +63,7 @@
 #include <linux/delayacct.h>
 #include <linux/reciprocal_div.h>
 #include <linux/unistd.h>
+#include <linux/pid_namespace.h>

 #include <asm/tlb.h>

@@ -1819,7 +1820,7 @@ 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.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./kernel/signal.c 2007-08-10 12:46:28.000000000 +0400
@@ -686,7 +686,7 @@ 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:
@@ -1084,7 +1084,7 @@ kill_proc_info(int sig, struct siginfo *
 {
  int error;
  rcu_read_lock();
- error = kill_pid_info(sig, info, find_pid(pid));
+ error = kill_pid_info(sig, info, find_vpid(pid));
  rcu_read_unlock();
```

```
  return error;
 }
@@ -1155,9 +1155,9 @@ 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));
+  ret = kill_pgrp_info(sig, info, find_vpid(-pid));
  } else {
-  ret = kill_pid_info(sig, info, find_pid(pid));
+  ret = kill_pid_info(sig, info, find_vpid(pid));
  }
  rcu_read_unlock();
  return ret;
@@ -1261,7 +1261,12 @@ EXPORT_SYMBOL(kill_pid);
 int
 kill_proc(pid_t pid, int sig, int priv)
 {
- return kill_proc_info(sig, __si_special(priv), pid);
+ int ret;
+
+ rcu_read_lock();
+ ret = kill_pid_info(sig, __si_special(priv), find_pid(pid));
+ rcu_read_unlock();
+ return ret;
 }

 /*
@@ -1439,7 +1444,22 @@ 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.
+  *
+  * the only it can is to switch its nsproxy with sys_unshare,
+  * bu uncharing pid namespaces is not allowed, so we'll always
+  * see relevant namespace
+  *
+  * write_lock() currently calls preempt_disable() which is the
+  * same as rcu_read_lock(), but according to Oleg, this is not
+  * correct to rely on this
+  */
+ rcu_read_lock();
+ info.si_pid = task_pid_nr_ns(tsk, tsk->parent->nsproxy->pid_ns);
+ rcu_read_unlock();
```

```
+
  info.si_uid = tsk->uid;

  /* FIXME: find out whether or not this is supposed to be c*time. */
@@ -1504,7 +1524,13 @@ static void do_notify_parent_cldstop(str

  info.si_signo = SIGCHLD;
  info.si_errno = 0;
- info.si_pid = tsk->pid;
+ /*
+  * see comment in do_notify_parent() abot the following 3 lines
+  */
+ rcu_read_lock();
+ info.si_pid = task_pid_nr_ns(tsk, tsk->parent->nsproxy->pid_ns);
+ rcu_read_unlock();
+
  info.si_uid = tsk->uid;

  /* FIXME: find out whether or not this is supposed to be c*time. */
@@ -1630,7 +1656,7 @@ 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.  */
@@ -1800,7 +1826,7 @@ 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;
    }

@@ -2189,7 +2215,7 @@ 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);
@@ -2205,12 +2231,12 @@ 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.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./kernel/sys.c 2007-08-10 12:43:21.000000000 +0400
@@ -151,7 +151,8 @@ 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)
@@ -159,7 +160,7 @@ asmlinkage long sys_setpriority(int whic
    break;
   case PRIO_PGRP:
    if (who)
-    pgrp = find_pid(who);
+    pgrp = find_vpid(who);
    else
     pgrp = task_pgrp(current);
    do_each_pid_task(pgrp, PIDTYPE_PGID, p) {
@@ -208,7 +209,8 @@ 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) {
@@ -219,7 +221,7 @@ asmlinkage long sys_getpriority(int whic
    break;
   case PRIO_PGRP:
```

```
    if (who)
-    pgrp = find_pid(who);
+    pgrp = find_vpid(who);
    else
     pgrp = task_pgrp(current);
    do_each_pid_task(pgrp, PIDTYPE_PGID, p) {
@@ -918,7 +920,7 @@ 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)
@@ -930,7 +932,7 @@ 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;

@@ -957,7 +959,8 @@ 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;
@@ -968,9 +971,12 @@ 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;
@@ -983,19 +989,20 @@ 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;
@@ -1007,7 +1014,7 @@ 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
@@ -1015,19 +1022,20 @@ 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;
@@ -1064,7 +1072,7 @@ asmlinkage long sys_setsid(void)
  group_leader->signal->tty = NULL;
  spin_unlock(&group_leader->sighand->siglock);

- err = task_pgrp_nr(group_leader);
+ err = task_pgrp_vnr(group_leader);
 out:
  write_unlock_irq(&tasklist_lock);
  return err;
--- ./kernel/sysctl.c.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./kernel/sysctl.c 2007-08-10 12:43:21.000000000 +0400
@@ -2325,7 +2325,7 @@ static int proc_do_cad_pid(ctl_table *ta
  pid_t tmp;
  int r;

- tmp = pid_nr(cad_pid);
+ tmp = pid_nr_ns(cad_pid, current->nsproxy->pid_ns);

  r = __do_proc_dointvec(&tmp, table, write, filp, buffer,
           lenp, ppos, NULL, NULL);
--- ./kernel/timer.c.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./kernel/timer.c 2007-08-10 12:43:21.000000000 +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.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./net/core/scm.c 2007-08-10 12:43:21.000000000 +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.ve18 2007-08-10 12:40:59.000000000 +0400
+++ ./net/unix/af_unix.c 2007-08-10 12:43:21.000000000 +0400
@@ -482,7 +482,7 @@ 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;
@@ -1129,7 +1129,7 @@ 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);
@@ -1190,7 +1190,7 @@ 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;
```

Subject: Re: [PATCH 19/20] Changes to show virtual ids to user
Posted by Andrew Morton on Tue, 14 Aug 2007 20:12:02 GMT
View Forum Message <> Reply to Message

On Fri, 10 Aug 2007 15:48:28 +0400
xemul@openvz.org wrote:

> 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.
>
> ...
>
> - si.si_pid = current->pid;
> + si.si_pid = task_pid_vnr(current);

This is going to be an ongoing maintenance problem: people will sneak

new references to current->pid into the tree and nobody will notice.

It'd be best to rename task_struct.pid to something else to catch such
problems and to force people to use the right accessors. Is that feasible?

Generally this is a tactic which should be used whenever things like this
are virtualised.

Subject: Re: [PATCH 19/20] Changes to show virtual ids to user
Posted by serue on Thu, 16 Aug 2007 12:13:54 GMT
View Forum Message <> Reply to Message

Quoting Andrew Morton (akpm@linux-foundation.org):
> On Fri, 10 Aug 2007 15:48:28 +0400
> xemul@openvz.org wrote:
>
> > 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.
> >
> > ...
> >
> > - si.si_pid = current->pid;
> > + si.si_pid = task_pid_vnr(current);
>
> This is going to be an ongoing maintenance problem: people will sneak
> new references to current->pid into the tree and nobody will notice.
>
> It'd be best to rename task_struct.pid to something else to catch such
> problems and to force people to use the right accessors. Is that feasible?

It's certainly feasible, and something we'd previously done for instance
in http://marc.info/?l=linux-kernel&m=113751118609597&w=2

> Generally this is a tactic which should be used whenever things like this
> are virtualised.

Ok, it's a big invasive patchset, but there's no reason we can't do it.

thanks,
-serge

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