Subject: [PATCH 0/28] Pid namespaces (two models)
Posted by Pavel Emelianov on Fri, 15 Jun 2007 15:55:43 GMT
View Forum Message <> Reply to Message

Long ago Sukadev and I sent two approaches for pid namespaces - the
hierarchical model in which namespaces are nested into each other,
and the flat model, where pids have only two values and creation of
level 3 namespace is prohibited.

After that I showed that multilevel model introduces a noticeable
overhead of approximately 1-2% to kernel standard operations like
fork() and getpid(). At the same time flat model showed no performance
hit on these tests.

Nevertheless multilevel model is worth living.

This set introduces booth models each under its config option. The
set is logically splitted into the following parts:
* [PREP] subset - the preparations for the namespaces. These patches
  by their own do not change the kernel behavior, but prepare the ground
  for the pid namespaces. This subset weights 14 patches;
* [FLAT] subset - the flat namespaces model. This is 6 patches;
* [MULTI] subset - the multilevel models. This is 6 patches also;
* Patch for cloning the namespace;
* Patch with Kconfig options.

The following tests were run:
[1] nptl perf test
[2] getpid() speed
[3] ltp (not for speed, but for kernel API checks)

The testing results summary:
* Flat model provides zero overhead in init namespace for all the
tests and less than 7% in the namespace for nptl test only.
* Multilevel model provides up to 2% overhead in init namespace and
more than 10% for nptl test in the level 2 namespace.

Testing details:

	perf, s	getpid
after unshare		

--------------------+-------------+-----------+

[1] ./perf -s 1000000 -t 1 -r 0 -T --sync-join
    nptl/perf.c from glibc-2.5

    error is 3 standard deviations

[2] getpid(2) done 10^9 times real time as reported by time(1)

The patches are for 2.6.22-rc4-mm2 tree.

Thanks,
Pavel

Subject: [PATCH 1/28] [PREP 1/14] Round up the API
Posted by Pavel Emelianov on Fri, 15 Jun 2007 15:59:18 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 inod
  struct autofs_sb_info *sbi = autofs_sbi(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->sighand->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 2/28] [PREP 2/14] Helpers to obtain pid numbers
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:00:23 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>

```
 pid.h   |   26 +++++++++++++
 sched.h |  127 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++------
 2 files changed, 143 insertions(+), 10 deletions(-)

--- ./include/linux/pid.h.nrhelpers 2007-06-15 14:31:19.000000000 +0400
+++ ./include/linux/pid.h 2007-06-15 14:39:27.000000000 +0400
@@ -98,6 +98,20 @@ extern struct pid *find_ge_pid(int nr);
 extern struct pid *alloc_pid(void);
```

```
 extern void FASTCALL(free_pid(struct pid *pid));

+struct pid_namespace;
+
+/*
+ * 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
+ */
+#ifndef CONFIG_PID_NS
 static inline pid_t pid_nr(struct pid *pid)
 {
  pid_t nr = 0;
@@ -106,6 +120,18 @@ static inline pid_t pid_nr(struct pid *p
  return nr;
 }

+static inline pid_t pid_vnr(struct pid *pid)
+{
+ return pid_nr(pid);
+}
+
+static inline pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
+{
+ return pid_nr(pid);
+}
+#else
+#endif
+
 #define do_each_pid_task(pid, type, task)    \
  do {        \
   struct hlist_node *pos___;    \
--- ./include/linux/sched.h.nrhelpers 2007-06-15 14:31:32.000000000 +0400
+++ ./include/linux/sched.h 2007-06-15 14:46:27.000000000 +0400
@@ -1153,16 +1153,6 @@ 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 +1178,123 @@ 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
+ */
+
+#ifndef CONFIG_PID_NS
+static inline pid_t task_pid_nr(struct task_struct *tsk)
+{
+ return tsk->pid;
+}
+
+static inline pid_t task_pid_vnr(struct task_struct *tsk)
+{
+ return task_pid_nr(tsk);
+}
+
```

```
+static inline pid_t task_pid_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ return task_pid_nr(tsk);
+}
+
+static inline void set_task_vpid(struct task_struct *tsk, pid_t nr)
+{
+}
+
+
+static inline pid_t task_tgid_nr(struct task_struct *tsk)
+{
+ return tsk->tgid;
+}
+
+static inline pid_t task_tgid_vnr(struct task_struct *tsk)
+{
+ return task_pid_nr(tsk);
+}
+
+static inline pid_t task_tgid_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ return task_pid_nr(tsk);
+}
+
+static inline void set_task_vtgid(struct task_struct *tsk, pid_t nr)
+{
+}
+
+
+static inline pid_t task_pgrp_nr(struct task_struct *tsk)
+{
+ return tsk->signal->pgrp;
+}
+
+static inline pid_t task_pgrp_vnr(struct task_struct *tsk)
+{
+ return task_pgrp_nr(tsk);
+}
+
+static inline pid_t task_pgrp_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ return task_pgrp_nr(tsk);
+}
+
```

```
+static inline void set_task_vpgrp(struct task_struct *tsk, pid_t nr)
+{
+}
+
+
+static inline pid_t task_session_nr(struct task_struct *tsk)
+{
+ return tsk->signal->__session;
+}
+
+static inline pid_t task_session_vnr(struct task_struct *tsk)
+{
+ return task_session_nr(tsk);
+}
+
+static inline pid_t task_session_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ return task_session_nr(tsk);
+}
+
+static inline void set_task_vsession(struct task_struct *tsk, pid_t nr)
+{
+}
+
+
+static inline pid_t task_ppid_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ return rcu_dereference(tsk->real_parent)->tgid;
+}
+#else
+#endif
+
 /**
  * pid_alive - check that a task structure is not stale
  * @p: Task structure to be checked.
```


Subject: [PATCH 3/28] [PREP 3/14] Helpers to find the task by its numerical ids
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:01:25 GMT
View Forum Message <> Reply to Message

When searching the task by numerical id on 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>

---

 include/linux/pid.h | 14 ++++++++++++-
 include/linux/sched.h | 31 ++++++++++++++++++++++++++++---
 kernel/pid.c | 13 +++++++------
 3 files changed, 49 insertions(+), 9 deletions(-)

--- ./include/linux/pid.h.findhelpers 2007-06-15 14:49:47.000000000 +0400
+++ ./include/linux/pid.h 2007-06-15 14:51:56.000000000 +0400
@@ -83,11 +83,23 @@ 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.
+ *
+ * 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() finr 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.
--- ./include/linux/sched.h.findhelpers 2007-06-15 14:49:47.000000000 +0400
+++ ./include/linux/sched.h 2007-06-15 14:55:39.000000000 +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.findhelpers 2007-06-15 14:48:58.000000000 +0400
+++ ./kernel/pid.c 2007-06-15 14:50:02.000000000 +0400
@@ -235,7 +235,7 @@ 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;
@@ -247,7 +247,7 @@ struct pid * fastcall find_pid(int nr)
  }
  return NULL;
 }
-EXPORT_SYMBOL_GPL(find_pid);
```

```
+EXPORT_SYMBOL_GPL(find_pid_ns);

 /*
  * attach_pid() must be called with the tasklist_lock write-held.
@@ -307,12 +307,13 @@ 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)
 {
@@ -339,7 +340,7 @@ 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;
```

Subject: [PATCH 4/28] [PREP 4/14] Make find_ge_pid() operate on virtual pids

Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:02:49 GMT

View Forum Message <> Reply to Message

find_ge_pid() is used in proc readdir and thus must be able to work with
virtual pids as well. The numerical ids are shown depending of what
namespace owns this proc mount. It assumes that the namespace in question
is stored in the superblock's private data.

Proc support will come later in this set, but this patch is logically tied
to the previous ones and git bisect safe, so it goes here.

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

---

 fs/proc/base.c | 15 +++++++++------
 include/linux/pid.h | 2 +-
 kernel/pid.c | 6 +++---
 3 files changed, 13 insertions(+), 10 deletions(-)

--- ./fs/proc/base.c.findgepid 2007-06-15 15:00:32.000000000 +0400
+++ ./fs/proc/base.c 2007-06-15 15:00:59.000000000 +0400
@@ -2291,7 +2291,8 @@ 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;
@@ -2299,9 +2300,9 @@ static struct task_struct *next_tgid(uns
  rcu_read_lock();
 retry:
  task = NULL;
- pid = find_ge_pid(tgid);
+ 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
@@ -2341,6 +2342,7 @@ 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;
@@ -2351,11 +2353,12 @@ 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);
--- ./include/linux/pid.h.findgepid 2007-06-15 15:00:44.000000000 +0400
+++ ./include/linux/pid.h 2007-06-15 15:00:59.000000000 +0400
@@ -105,7 +105,7 @@ extern struct pid *FASTCALL(find_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));
--- ./kernel/pid.c.findgepid 2007-06-15 15:00:44.000000000 +0400
+++ ./kernel/pid.c 2007-06-15 15:00:59.000000000 +0400
@@ -351,15 +351,15 @@ struct pid *find_get_pid(pid_t nr)
  *
  * 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;
```

Subject: [PATCH 5/28] [PREP 5/14] Split INIT_STRUCT_PID into two parts
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:03:37 GMT
View Forum Message <> Reply to Message

Virtual pids will have their own meanings of pid numerical ids, so
when initializing init_pid we need to split the data referring to
the struct pid itself and to the ids of the pid

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

---

 init_task.h | 13 ++++++++++---
 1 files changed, 10 insertions(+), 3 deletions(-)

diff --git a/include/linux/init_task.h b/include/linux/init_task.h
index 276ccaa..ef10654 100644
--- a/include/linux/init_task.h
+++ b/include/linux/init_task.h
@@ -89,17 +89,24 @@ extern struct nsproxy init_nsproxy;

 extern struct group_info init_groups;

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

 #define INIT_PID_LINK(type)   \

_____
Containers mailing list

Subject: [PATCH 6/28] [PREP 6/14] Changes to show virtual ids to user
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:04:29 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                   |    7 +++++--
 fs/proc/array.c             |   21 +++++++++++++-------
 fs/proc/base.c              |   23 +++++++++++++++--------
 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               |   15 ++++++++++----
 kernel/futex.c              |   23 +++++++++++++++--------
 kernel/ptrace.c             |    4 +++-
 kernel/sched.c              |    3 ++-
 kernel/signal.c             |   42 ++++++++++++++++++++++++++++++++----------
 kernel/sys.c                |   41 ++++++++++++++++++++++++++++++++---------
 kernel/timer.c              |    7 ++++---
```

```
 net/core/scm.c             |    4 +++-
 net/unix/af_unix.c         |    6 +++---
 24 files changed, 193 insertions(+), 120 deletions(-)

--- ./arch/ia64/kernel/signal.c.pidnshooks 2007-06-15 15:00:32.000000000 +0400
+++ ./arch/ia64/kernel/signal.c 2007-06-15 15:02:29.000000000 +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.pidnshooks 2007-06-15 15:00:32.000000000 +0400
+++ ./arch/parisc/kernel/signal.c 2007-06-15 15:02:29.000000000 +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.pidnshooks 2007-06-15 15:00:44.000000000 +0400
+++ ./drivers/char/tty_io.c 2007-06-15 15:02:29.000000000 +0400
@@ -103,6 +103,7 @@
 #include <linux/selection.h>

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

 #undef TTY_DEBUG_HANGUP

@@ -3062,7 +3063,7 @@ 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);
 }

 /**
@@ -3096,7 +3097,7 @@ 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;
@@ -3133,7 +3134,7 @@ 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.pidnshooks 2007-06-15 15:00:44.000000000 +0400
+++ ./fs/binfmt_elf.c 2007-06-15 15:02:29.000000000 +0400
@@ -1392,10 +1392,10 @@ 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
@@ -1438,10 +1438,10 @@ 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.pidnshooks 2007-06-15 15:00:44.000000000 +0400
+++ ./fs/binfmt_elf_fdpic.c 2007-06-15 15:02:29.000000000 +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.pidnshooks 2007-06-15 15:00:32.000000000 +0400
+++ ./fs/exec.c 2007-06-15 15:02:29.000000000 +0400
@@ -716,6 +716,9 @@ static int de_thread(struct task_struct
   attach_pid(tsk, PIDTYPE_PID,  find_pid(tsk->pid));
   transfer_pid(leader, tsk, PIDTYPE_PGID);
   transfer_pid(leader, tsk, PIDTYPE_SID);
+  set_task_vpgrp(leader, task_pid_vnr(current));
+  set_task_vpid(leader, task_pid_vnr(current));
```

```
+  set_task_vtgid(current, task_pid_vnr(current));
   list_replace_rcu(&leader->tasks, &tsk->tasks);

   tsk->group_leader = tsk;
@@ -1304,7 +1307,7 @@ 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;
@@ -1376,7 +1379,7 @@ 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/proc/array.c.pidnshooks 2007-06-15 15:00:44.000000000 +0400
+++ ./fs/proc/array.c 2007-06-15 15:02:29.000000000 +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);

@@ -382,6 +388,7 @@ 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);
@@ -414,9 +421,9 @@ 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);
  }
@@ -447,7 +454,7 @@ 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, current->nsproxy->pid_ns),
   tcomm,
   state,
   ppid,
--- ./fs/proc/base.c.pidnshooks 2007-06-15 15:00:59.000000000 +0400
+++ ./fs/proc/base.c 2007-06-15 15:02:29.000000000 +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:
@@ -1907,14 +1908,14 @@ 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));
 }

@@ -2264,6 +2265,7 @@ 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)
@@ -2273,8 +2275,9 @@ 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();
@@ -2490,6 +2493,7 @@ 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;
@@ -2498,8 +2502,9 @@ 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();
@@ -2530,14 +2535,14 @@ 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;
  }
@@ -2606,6 +2611,7 @@ 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)
@@ -2639,12 +2645,13 @@ 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.pidnshooks 2007-06-15 15:00:32.000000000 +0400
+++ ./include/net/scm.h 2007-06-15 15:02:29.000000000 +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.pidnshooks 2007-06-15 15:00:32.000000000 +0400
+++ ./ipc/mqueue.c 2007-06-15 15:02:29.000000000 +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.pidnshooks 2007-06-15 15:00:32.000000000 +0400
+++ ./ipc/msg.c 2007-06-15 15:02:29.000000000 +0400
@@ -615,7 +615,7 @@ 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();
@@ -699,7 +699,7 @@ 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)) {
@@ -814,7 +814,7 @@ 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.pidnshooks 2007-06-15 15:00:32.000000000 +0400
+++ ./ipc/sem.c 2007-06-15 15:02:29.000000000 +0400
@@ -797,7 +797,7 @@ 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);
@@ -1200,7 +1200,7 @@ 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);
@@ -1215,7 +1215,7 @@ 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)
@@ -1386,7 +1386,7 @@ 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.pidnshooks 2007-06-15 15:00:32.000000000 +0400
+++ ./ipc/shm.c 2007-06-15 15:02:29.000000000 +0400
@@ -170,7 +170,7 @@ 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);
 }
@@ -215,7 +215,7 @@ 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.pidnshooks 2007-06-15 15:00:32.000000000 +0400
+++ ./kernel/capability.c 2007-06-15 15:02:29.000000000 +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.pidnshooks 2007-06-15 15:00:44.000000000 +0400
+++ ./kernel/exit.c 2007-06-15 15:02:29.000000000 +0400
@@ -1050,15 +1050,17 @@ 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;
   }

@@ -1129,9 +1131,12 @@ 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;
@@ -1249,7 +1254,7 @@ 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) {
@@ -1257,7 +1262,7 @@ 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.  */
@@ -1295,6 +1300,7 @@ 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;
@@ -1313,11 +1319,12 @@ 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;

@@ -1388,11 +1395,11 @@ 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);
@@ -1412,6 +1419,7 @@ 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;
@@ -1429,7 +1437,8 @@ 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);
@@ -1440,7 +1449,7 @@ 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.pidnshooks 2007-06-15 15:00:44.000000000 +0400
+++ ./kernel/fork.c 2007-06-15 15:02:29.000000000 +0400
@@ -49,6 +49,7 @@
 #include <linux/delayacct.h>
 #include <linux/taskstats_kern.h>
 #include <linux/random.h>
+#include <linux/pid.h>

 #include <asm/pgtable.h>
 #include <asm/pgalloc.h>
@@ -935,7 +936,7 @@ asmlinkage long sys_set_tid_address(int
 {
  current->clear_child_tid = tidptr;

- return current->pid;
+ return task_pid_vnr(current);
 }

 static inline void rt_mutex_init_task(struct task_struct *p)
@@ -1030,6 +1031,7 @@ static struct task_struct *copy_process(
  delayacct_tsk_init(p); /* Must remain after dup_task_struct() */
  copy_flags(clone_flags, p);
  p->pid = pid_nr(pid);
+ set_task_vpid(p, pid_vnr(pid));
  INIT_LIST_HEAD(&p->children);
  INIT_LIST_HEAD(&p->sibling);
  p->vfork_done = NULL;
@@ -1105,8 +1107,11 @@ static struct task_struct *copy_process(
 #endif

  p->tgid = p->pid;
- if (clone_flags & CLONE_THREAD)
+ set_task_vtgid(p, task_pid_vnr(p));
+ if (clone_flags & CLONE_THREAD) {
    p->tgid = current->tgid;
+  set_task_vtgid(p, task_pid_vnr(current));
+ }
```

```
  if ((retval = security_task_alloc(p)))
    goto bad_fork_cleanup_policy;
@@ -1261,6 +1266,8 @@ static struct task_struct *copy_process(
     p->signal->tty = current->signal->tty;
     p->signal->pgrp = task_pgrp_nr(current);
     set_task_session(p, task_session_nr(current));
+    set_task_vpgrp(p, task_pgrp_vnr(current));
+    set_task_vsession(p, task_session_vnr(current));
     attach_pid(p, PIDTYPE_PGID, task_pgrp(current));
     attach_pid(p, PIDTYPE_SID, task_session(current));

@@ -1280,7 +1287,7 @@ 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;
@@ -1382,7 +1389,7 @@ long do_fork(unsigned long clone_flags,

   if (!pid)
     return -EAGAIN;
- nr = pid->nr;
+ nr = pid_vnr(pid);
   if (unlikely(current->ptrace)) {
     trace = fork_traceflag (clone_flags);
     if (trace)
--- ./kernel/futex.c.pidnshooks 2007-06-15 15:00:32.000000000 +0400
+++ ./kernel/futex.c 2007-06-15 15:02:29.000000000 +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"
@@ -629,7 +631,7 @@ static int wake_futex_pi(u32 __user *uad
    * preserve the owner died bit.)
    */
   if (!(uval & FUTEX_OWNER_DIED)) {
-    newval = FUTEX_WAITERS | new_owner->pid;
+    newval = FUTEX_WAITERS | task_pid_vnr(new_owner);
     /* Keep the FUTEX_WAITER_REQUEUED flag if it was set */
```

```
   newval |= (uval & FUTEX_WAITER_REQUEUED);

@@ -1345,7 +1347,7 @@ static int fixup_pi_state_owner(u32 __us
     struct futex_hash_bucket *hb,
     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;
@@ -1731,7 +1733,7 @@ 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);

  pagefault_disable();
  curval = futex_atomic_cmpxchg_inatomic(uaddr, 0, newval);
@@ -1741,7 +1743,7 @@ static int futex_lock_pi(u32 __user *uad
   goto uaddr_faulted;

  /* We own the lock already */
- if (unlikely((curval & FUTEX_TID_MASK) == current->pid)) {
+ if (unlikely((curval & FUTEX_TID_MASK) == task_pid_vnr(current))) {
   if (!detect && 0)
    force_sig(SIGKILL, current);
   /*
@@ -1771,7 +1773,7 @@ static int futex_lock_pi(u32 __user *uad
   */
  if ((curval & FUTEX_WAITER_REQUEUED) && !(curval & FUTEX_TID_MASK)) {
   /* set current as futex owner */
-  newval = curval | current->pid;
+  newval = curval | task_pid_vnr(current);
   lock_held = 1;
  } else
   /* Set the WAITERS flag, so the owner will know it has someone
@@ -1812,7 +1814,7 @@ static int futex_lock_pi(u32 __user *uad
    */
   if (curval & FUTEX_OWNER_DIED) {
    uval = newval;
-   newval = current->pid |
+   newval = task_pid_vnr(current) |
     FUTEX_OWNER_DIED | FUTEX_WAITERS;

    pagefault_disable();
@@ -1971,7 +1973,7 @@ 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:
@@ -1994,7 +1996,8 @@ retry_unlocked:
   */
  if (!(uval & FUTEX_OWNER_DIED)) {
   pagefault_disable();
-  uval = futex_atomic_cmpxchg_inatomic(uaddr, current->pid, 0);
+  uval = futex_atomic_cmpxchg_inatomic(uaddr,
+    task_pid_vnr(current), 0);
   pagefault_enable();
  }

@@ -2004,7 +2007,7 @@ 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;

  /*
@@ -2275,7 +2278,7 @@ 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.pidnshooks 2007-06-15 15:00:32.000000000 +0400
+++ ./kernel/ptrace.c 2007-06-15 15:02:29.000000000 +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.pidnshooks 2007-06-15 15:00:32.000000000 +0400
+++ ./kernel/sched.c 2007-06-15 15:02:29.000000000 +0400
@@ -60,6 +60,7 @@
 #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>
@@ -1459,7 +1460,7 @@ 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.pidnshooks 2007-06-15 15:00:44.000000000 +0400
+++ ./kernel/signal.c 2007-06-15 15:02:29.000000000 +0400
@@ -674,7 +674,7 @@ 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:
@@ -1124,13 +1124,15 @@ 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 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 +1147,12 @@ 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);
+   ret = kill_pgrp_info(sig, info, pid);
  } else {
-   ret = kill_pid_info(sig, info, find_pid(pid));
+   pid = find_vpid(pid_nr);
+   ret = kill_pid_info(sig, info, pid);
  }
  rcu_read_unlock();
  return ret;
@@ -1430,7 +1434,11 @@ 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. */
@@ -1495,7 +1503,11 @@ 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. */
@@ -1624,7 +1636,7 @@ 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.  */
@@ -1794,7 +1806,7 @@ 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;
    }

@@ -2183,7 +2195,7 @@ 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);
@@ -2199,12 +2211,12 @@ 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.pidnshooks 2007-06-15 15:00:44.000000000 +0400
+++ ./kernel/sys.c 2007-06-15 15:02:29.000000000 +0400
```

```
@@ -674,7 +674,8 @@ 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)
@@ -731,7 +732,8 @@ 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) {
@@ -1436,7 +1438,7 @@ 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)
@@ -1448,7 +1450,7 @@ 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;

@@ -1475,7 +1477,8 @@ 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;
```

```
@@ -1486,9 +1489,13 @@ 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);
+  set_task_vpgrp(p, pid_vnr(pid));
  }

  err = 0;
@@ -1501,19 +1508,20 @@ 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;
@@ -1525,7 +1533,7 @@ 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
@@ -1533,19 +1541,20 @@ 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;
@@ -1577,12 +1586,14 @@ asmlinkage long sys_setsid(void)

  group_leader->signal->leader = 1;
  __set_special_pids(session, session);
+ set_task_vsession(current, task_pid_vnr(group_leader));
+ set_task_vpgrp(current, task_pid_vnr(group_leader));

  spin_lock(&group_leader->sighand->siglock);
  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/timer.c.pidnshooks 2007-06-15 15:00:32.000000000 +0400
+++ ./kernel/timer.c 2007-06-15 15:02:29.000000000 +0400
@@ -36,6 +36,7 @@
 #include <linux/delay.h>
 #include <linux/tick.h>
 #include <linux/kallsyms.h>
+#include <linux/pid_namespace.h>
```

```
 #include <asm/uaccess.h>
 #include <asm/unistd.h>
@@ -953,7 +954,7 @@ asmlinkage unsigned long sys_alarm(unsig
  */
 asmlinkage long sys_getpid(void)
 {
- return current->tgid;
+ return task_tgid_vnr(current);
 }

 /*
@@ -967,7 +968,7 @@ 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;
@@ -1099,7 +1100,7 @@ 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.pidnshooks 2007-06-15 15:00:32.000000000 +0400
+++ ./net/core/scm.c 2007-06-15 15:02:29.000000000 +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.pidnshooks 2007-06-15 15:00:32.000000000 +0400
+++ ./net/unix/af_unix.c 2007-06-15 15:02:29.000000000 +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;
```

Subject: [PATCH 7/28] [PREP 7/14] Prepare proc_flust_task to flush entries from multiple proc trees

Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:05:32 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          |    2 +-
 3 files changed, 27 insertions(+), 6 deletions(-)

--- ./fs/proc/base.c.procflushtask 2007-06-15 15:02:29.000000000 +0400
+++ ./fs/proc/base.c 2007-06-15 15:04:18.000000000 +0400
@@ -2184,7 +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];
@@ -2192,7 +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);
@@ -2204,7 +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;

@@ -2230,6 +2230,25 @@ 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.
+ */
+#ifndef CONFIG_PID_NS
```

```
+static inline void proc_flush_task_ns(struct task_struct *tsk, struct pid *pid)
+{
+}
+#else
+#endif
+
+void proc_flush_task(struct task_struct *task, struct pid *pid)
+{
+ proc_flush_task_mnt(task, proc_mnt);
+ if (pid != NULL)
+  proc_flush_task_ns(task, pid);
+}
+
 static struct dentry *proc_pid_instantiate(struct inode *dir,
        struct dentry * dentry,
        struct task_struct *task, const void *ptr)
--- ./include/linux/proc_fs.h.procflushtask 2007-06-15 15:00:32.000000000 +0400
+++ ./include/linux/proc_fs.h 2007-06-15 15:03:10.000000000 +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 +223,9 @@ 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; }
--- ./kernel/exit.c.procflushtask 2007-06-15 15:02:29.000000000 +0400
+++ ./kernel/exit.c 2007-06-15 15:03:10.000000000 +0400
@@ -185,7 +185,7 @@ repeat:
  }

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

Subject: [PATCH 8/28] [PREP 8/14] Make proc be able to have multiple super blocks
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:06:30 GMT
View Forum Message <> Reply to Message

Each namespace must have its own proc super block where dentries
and inodes representing the tasks live. Plus proc super block must
hold the namespace it draws the pids from.

Making one proc mount with filtering looks very ugly for the following
reasons. Two namespaces can have one numerical id refer to different
tasks, e.g. 1 in init namespace refers to init, but 1 in sub namespace
does not. This creates difficulties in proc. However having many proc
trees solves this problem and prepares the ground for the proc
virtualization.

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

```
 fs/proc/inode.c         |   20 +++++++++---
 fs/proc/internal.h      |    2 +
 fs/proc/root.c          |   75 ++++++++++++++++++++++++++++++++++++++++++++++++--
 include/linux/proc_fs.h |    3 +
 4 files changed, 91 insertions(+), 9 deletions(-)

--- ./fs/proc/inode.c.procmount 2007-06-15 15:00:32.000000000 +0400
+++ ./fs/proc/inode.c 2007-06-15 15:04:52.000000000 +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");
--- ./fs/proc/internal.h.procmount 2007-06-15 15:00:32.000000000 +0400
+++ ./fs/proc/internal.h 2007-06-15 15:04:52.000000000 +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);
--- ./fs/proc/root.c.procmount 2007-06-15 15:00:32.000000000 +0400
+++ ./fs/proc/root.c 2007-06-15 15:05:16.000000000 +0400
```

```
@@ -18,32 +18,79 @@
 #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;
+ get_pid_ns(ns);
+ sb->s_fs_info = data;
+ 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);
+
+ 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;
+ }
+
+ return simple_set_mnt(mnt, sb);
+}
+
+static void proc_kill_sb(struct super_block *sb)
+{
+ put_pid_ns((struct pid_namespace *)sb->s_fs_info);
+ kill_anon_super(sb);
 }

 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)
@@ -153,6 +200,28 @@ 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;
+}
+
 EXPORT_SYMBOL(proc_symlink);
 EXPORT_SYMBOL(proc_mkdir);
 EXPORT_SYMBOL(create_proc_entry);
--- ./include/linux/proc_fs.h.procmount 2007-06-15 15:03:10.000000000 +0400
+++ ./include/linux/proc_fs.h 2007-06-15 15:04:52.000000000 +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 *);

 /*
```


Subject: [PATCH 9/28] [PREP 9/14] Masquerade the siginfo when sending a pid to a foreign namespace

Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:07:23 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 | 18 ++++++++++++++++++
 2 files changed, 28 insertions(+)

--- ./include/linux/pid.h.sigmasqerade 2007-06-15 15:00:59.000000000 +0400
+++ ./include/linux/pid.h 2007-06-15 15:08:39.000000000 +0400
@@ -141,6 +141,16 @@ static inline pid_t pid_nr_ns(struct pid
 {
  return pid_nr(pid);
 }
+
+/*
+ * 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 1;
+}
 #else
 #endif

--- ./kernel/signal.c.sigmasqerade 2007-06-15 15:02:29.000000000 +0400
+++ ./kernel/signal.c 2007-06-15 15:06:04.000000000 +0400
@@ -1124,6 +1124,22 @@ EXPORT_SYMBOL_GPL(kill_pid_info_as_uid);
  * is probably wrong. Should make it like BSD or SYSV.
  */

+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;
@@ -1149,9 +1165,11 @@ static int kill_something_info(int sig,
   ret = count ? retval : -ESRCH;
  } 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 {
   pid = find_vpid(pid_nr);
+  masquerade_siginfo(current->nsproxy->pid_ns, pid, info);
   ret = kill_pid_info(sig, info, pid);
  }
  rcu_read_unlock();
```

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

---

Subject: [PATCH 10/28] [PREP 10/14] Prepare some pid.c routines for the namespaces support
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:08:23 GMT

View Forum Message <> Reply to Message

The alloc_pid(), free_pid() and put_pid() are splitted into two parts
 - the generic pid manipulation, like refcount and task hlist heads;
 - manipulations with pids numerical values.

E.g. alloc pid allocates the struct pid and initializes its refcount etc and
calls the alloc_pid_nrs() to allocate the pid numerical ids and hash it into
the hash-tables.

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

```
 pid.c |   90 ++++++++++++++++++++++++++++++++++++++++++++++++++-------------------------
 1 files changed, 56 insertions(+), 34 deletions(-)

--- ./kernel/pid.c.pidsplit 2007-06-14 15:57:41.000000000 +0400
+++ ./kernel/pid.c 2007-06-14 16:34:28.000000000 +0400
@@ -174,13 +174,64 @@ static int next_pidmap(struct pid_namesp
  return -1;
```

```
 }

+#ifndef CONFIG_PID_NS
+static inline int alloc_pid_nrs(struct pid *pid)
+{
+ int nr;
+
+ nr = alloc_pidmap(&init_pid_ns);
+ if (nr < 0)
+  return nr;
+
+ pid->nr = nr;
+ spin_lock_irq(&pidmap_lock);
+ hlist_add_head_rcu(&pid->pid_chain, &pid_hash[pid_hashfn(nr)]);
+ spin_unlock_irq(&pidmap_lock);
+ return 0;
+}
+
+static inline void unhash_pid_nrs(struct pid *pid)
+{
+ /* We can be called with write_lock_irq(&tasklist_lock) held */
+ unsigned long flags;
+
+ spin_lock_irqsave(&pidmap_lock, flags);
+ hlist_del_rcu(&pid->pid_chain);
+ spin_unlock_irqrestore(&pidmap_lock, flags);
+
+ free_pidmap(&init_pid_ns, pid->nr);
+}
+
+static inline void free_pid_nrs(struct pid *pid)
+{
+}
+
+struct pid * fastcall find_pid_ns(int nr, struct pid_namespace *ns)
+{
+ struct hlist_node *elem;
+ struct pid *pid;
+
+ hlist_for_each_entry_rcu(pid, elem,
+    &pid_hash[pid_hashfn(nr)], pid_chain) {
+  if (pid->nr == nr)
+   return pid;
+ }
+ return NULL;
+}
+#else
+#endif
```

```
+
+EXPORT_SYMBOL_GPL(find_pid_ns);
+
 fastcall void put_pid(struct pid *pid)
 {
  if (!pid)
   return;
  if ((atomic_read(&pid->count) == 1) ||
-      atomic_dec_and_test(&pid->count))
+      atomic_dec_and_test(&pid->count)) {
+  free_pid_nrs(pid);
   kmem_cache_free(pid_cachep, pid);
+ }
 }
 EXPORT_SYMBOL_GPL(put_pid);

@@ -192,14 +243,7 @@ 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 */
- unsigned long flags;
-
- spin_lock_irqsave(&pidmap_lock, flags);
- hlist_del_rcu(&pid->pid_chain);
- spin_unlock_irqrestore(&pidmap_lock, flags);
-
- free_pidmap(&init_pid_ns, pid->nr);
+ unhash_pid_nrs(pid);
  call_rcu(&pid->rcu, delayed_put_pid);
 }

@@ -207,47 +251,25 @@ struct pid *alloc_pid(void)
 {
  struct pid *pid;
  enum pid_type type;
- int nr = -1;

  pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
  if (!pid)
   goto out;

- nr = alloc_pidmap(current->nsproxy->pid_ns);
- if (nr < 0)
-  goto out_free;
-
  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)]);
- spin_unlock_irq(&pidmap_lock);
+ if (alloc_pid_nrs(pid))
+  goto out_free;

-out:
  return pid;

 out_free:
  kmem_cache_free(pid_cachep, pid);
- pid = NULL;
- goto out;
-}
-
-struct pid * fastcall find_pid_ns(int nr, struct pid_namespace *ns)
-{
- struct hlist_node *elem;
- struct pid *pid;
-
- hlist_for_each_entry_rcu(pid, elem,
-   &pid_hash[pid_hashfn(nr)], pid_chain) {
-  if (pid->nr == nr)
-   return pid;
- }
+out:
  return NULL;
 }
-EXPORT_SYMBOL_GPL(find_pid_ns);

 /*
  * attach_pid() must be called with the tasklist_lock write-held.
```


Subject: [PATCH 11/28] [PREP 11/14] Add support for multiple hash tables in pid.c
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:09:35 GMT
View Forum Message <> Reply to Message

It turned out that virtual pids require two hash tables for pid searching.
E.g. flat model hashed pid by its global and virtual ids, multilevel model
uses one hash to find the pid by number, and the other one to find the

number by the struct pid.

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

---

 pid.c | 18 ++++++++++++++++-
 1 files changed, 17 insertions(+), 1 deletion(-)

--- ./kernel/pid.c.pidmorecaches 2007-06-15 15:09:29.000000000 +0400
+++ ./kernel/pid.c 2007-06-15 15:11:14.000000000 +0400
@@ -54,6 +54,17 @@ static inline int mk_pid(struct pid_name
 #define find_next_offset(map, off)   \
   find_next_zero_bit((map)->page, BITS_PER_PAGE, off)

+#ifdef CONFIG_PID_NS
+/*
+ * pid namespaces will require the additional hash to store the
+ * pid-to-namespace relations. so declare it here and define a hash
+ * fun of two arguments - nr and the namespce
+ */
+static struct hlist_head *pid_hash2;
+#define pid_ehashfn(nr, ns) hash_long((unsigned long)nr + (unsigned long)ns, \
+   pidhash_shift)
+#endif
+
 /*
  * PID-map pages start out as NULL, they get allocated upon
  * first use and are never deallocated. This way a low pid_max
@@ -420,12 +431,17 @@ void __init pidhash_init(void)
  printk("PID hash table entries: %d (order: %d, %Zd bytes)\n",
   pidhash_size, pidhash_shift,
   pidhash_size * sizeof(struct hlist_head));
-
+#ifdef CONFIG_PID_NS
+ pidhash_size *= 2;
+#endif
  pid_hash = alloc_bootmem(pidhash_size * sizeof(*(pid_hash)));
  if (!pid_hash)
   panic("Could not alloc pidhash!\n");
  for (i = 0; i < pidhash_size; i++)
   INIT_HLIST_HEAD(&pid_hash[i]);
+#ifdef CONFIG_PID_NS
+ pid_hash2 = pid_hash + (pidhash_size / 2);
+#endif
 }

 void __init pidmap_init(void)

Subject: [PATCH 12/28] [PREP 12/14] Add proc vfsmount on struct pid_namespace
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:10:28 GMT
View Forum Message <> Reply to Message

Each namespace is supposed to have its own proc mount. So add the
struct vfsmount pointer to struct pid_namespace and introduce the
helpers to initialize it properly.

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

```
 fs/proc/root.c              |   20 ++++++++++++++++++++
 include/linux/pid_namespace.h |    3 +++
 include/linux/proc_fs.h       |   12 ++++++++++++
 3 files changed, 35 insertions(+)

--- ./fs/proc/root.c.pidnswithproc 2007-06-15 15:05:16.000000000 +0400
+++ ./fs/proc/root.c 2007-06-15 15:12:25.000000000 +0400
@@ -107,6 +107,9 @@ void __init proc_root_init(void)
   unregister_filesystem(&proc_fs_type);
   return;
  }
+#ifdef CONFIG_PID_NS
+ init_pid_ns.proc_mnt = proc_mnt;
+#endif
  proc_misc_init();
  proc_net = proc_mkdir("net", NULL);
  proc_net_stat = proc_mkdir("net/stat", NULL);
@@ -222,6 +225,23 @@ struct proc_dir_entry * create_proc_root
  return de;
 }

+int pid_ns_prepare_proc(struct pid_namespace *ns)
+{
+ struct vfsmount *mnt;
+
+ mnt = kern_mount(&proc_fs_type);
+ if (IS_ERR(mnt))
+  return PTR_ERR(mnt);
+
+ ns->proc_mnt = mnt;
```

```
+ return 0;
+}
+
+void pid_ns_release_proc(struct pid_namespace *ns)
+{
+ mntput(ns->proc_mnt);
+}
+
 EXPORT_SYMBOL(proc_symlink);
 EXPORT_SYMBOL(proc_mkdir);
 EXPORT_SYMBOL(create_proc_entry);
--- ./include/linux/pid_namespace.h.pidnswithproc 2007-06-15 15:00:32.000000000 +0400
+++ ./include/linux/pid_namespace.h 2007-06-15 15:12:25.000000000 +0400
@@ -20,6 +20,9 @@ struct pid_namespace {
  struct pidmap pidmap[PIDMAP_ENTRIES];
  int last_pid;
  struct task_struct *child_reaper;
+#ifdef CONFIG_PROC_FS
+ struct vfsmount *proc_mnt;
+#endif
 };

 extern struct pid_namespace init_pid_ns;
--- ./include/linux/proc_fs.h.pidnswithproc 2007-06-15 15:04:52.000000000 +0400
+++ ./include/linux/proc_fs.h 2007-06-15 15:12:25.000000000 +0400
@@ -144,6 +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
  */
@@ -251,6 +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)
```


Subject: [PATCH 13/28] [PREP 13/14] Miscellaneous preparations in pid namespaces

Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:11:28 GMT

View Forum Message <> Reply to Message

The most important one is moving exit_task_namespaces behind exit_notify
in do_exit() to make it possible to see the task's pid namespace to
properly notify the parent.

The other important change is redefining child_reaper(). It turned out
that all the places that use it pass current task and thus it is safe
to dereference task->nsproxy pointer.

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

```
 include/linux/pid_namespace.h |    7 ++++++-
 kernel/exit.c                 |    2 +-
 kernel/pid.c                  |    2 ++
 3 files changed, 9 insertions(+), 2 deletions(-)

--- ./include/linux/pid_namespace.h.nsmiscprep 2007-06-15 15:12:25.000000000 +0400
+++ ./include/linux/pid_namespace.h 2007-06-15 15:13:07.000000000 +0400
@@ -29,7 +29,9 @@ extern struct pid_namespace init_pid_ns;

 static inline void get_pid_ns(struct pid_namespace *ns)
 {
+#ifdef CONFIG_PID_NS
  kref_get(&ns->kref);
+#endif
 }

 extern struct pid_namespace *copy_pid_ns(int flags, struct pid_namespace *ns);
@@ -37,12 +39,15 @@ extern void free_pid_ns(struct kref *kre

 static inline void put_pid_ns(struct pid_namespace *ns)
 {
+#ifdef CONFIG_PID_NS
```

```
  kref_put(&ns->kref, free_pid_ns);
+#endif
 }

 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.nsmiscprep 2007-06-15 15:03:10.000000000 +0400
+++ ./kernel/exit.c 2007-06-15 15:13:07.000000000 +0400
@@ -955,8 +955,8 @@ fastcall NORET_TYPE void do_exit(long co

  tsk->exit_code = code;
  proc_exit_connector(tsk);
- exit_task_namespaces(tsk);
  exit_notify(tsk);
+ exit_task_namespaces(tsk);
 #ifdef CONFIG_NUMA
  mpol_free(tsk->mempolicy);
  tsk->mempolicy = NULL;
--- ./kernel/pid.c.nsmiscprep 2007-06-15 15:11:14.000000000 +0400
+++ ./kernel/pid.c 2007-06-15 15:13:07.000000000 +0400
@@ -82,6 +82,8 @@ 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).
```


Subject: [PATCH 14/28] [PREP 14/14] Hold the struct pid till flushing the proc trees in release_task()

Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:12:24 GMT

View Forum Message <> Reply to Message

When task is released its pids are detached first and then the proc is flushed.
With the namespaces we have to hold the pid till proc flush to have all the

namespaces to flush the dentries from.

This get_pid()/put_pid() pair introduces some performance overhead on nptl perf
test, so if the pid belongs to init namespace only, we can optimize this out.

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

```
 exit.c |   18 ++++++++++++++++-
 1 files changed, 17 insertions(+), 1 deletion(-)

--- ./kernel/exit.c.flushsave 2007-06-15 15:13:07.000000000 +0400
+++ ./kernel/exit.c 2007-06-15 15:13:45.000000000 +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 belogs 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, NULL);
```

```
+ 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 15/28] The namespace cloning
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:13:17 GMT
View Forum Message <> Reply to Message

This is the core of the set - the namespace cloning.

The cloning consists of two stages - creating of the new namespace and moving a task into it. Create and move is not good as the error path just puts the new namespaces and thus keep the task in it.

So after the new namespace is clones task still outside it. It is injected inside explicitly after all the operations that might fail are finished.

Another important thing is that task must be alone in its group and session and must not be splitted into threads. This is because all task's pids are moved to the new ns and if they are shared with someone else this someone may happen to be half-inserted into the new space.

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

```
 include/linux/pid_namespace.h |    2
 include/linux/sched.h         |    1
 kernel/fork.c                 |   15 +++-
 kernel/nsproxy.c              |    6 +
 kernel/pid.c                  |  152 ++++++++++++++++++++++++++++++++++++++++++++-
 5 files changed, 172 insertions(+), 4 deletions(-)

--- ./include/linux/pid_namespace.h.clonepidns 2007-06-15 15:13:07.000000000 +0400
+++ ./include/linux/pid_namespace.h 2007-06-15 15:14:33.000000000 +0400
@@ -50,4 +50,6 @@ static inline struct task_struct *child_
  return tsk->nsproxy->pid_ns->child_reaper;
 }

+int move_init_to_ns(struct task_struct *tsk, struct nsproxy *nsp);
+
```

```
 #endif /* _LINUX_PID_NS_H */
--- ./include/linux/sched.h.clonepidns 2007-06-15 15:00:44.000000000 +0400
+++ ./include/linux/sched.h 2007-06-15 15:14:33.000000000 +0400
@@ -25,6 +25,7 @@
 #define CLONE_STOPPED  0x02000000 /* Start in stopped state */
 #define CLONE_NEWUTS  0x04000000 /* New utsname group? */
 #define CLONE_NEWIPC  0x08000000 /* New ipcs */
+#define CLONE_NEWPIDS  0x10000000 /* New pids */

 /*
  * Scheduling policies
--- ./kernel/fork.c.clonepidns 2007-06-15 15:02:29.000000000 +0400
+++ ./kernel/fork.c 2007-06-15 15:18:15.000000000 +0400
@@ -1623,7 +1623,7 @@ asmlinkage long sys_unshare(unsigned lon
  err = -EINVAL;
  if (unshare_flags & ~(CLONE_THREAD|CLONE_FS|CLONE_NEWNS|CLONE_SIGHAND|
     CLONE_VM|CLONE_FILES|CLONE_SYSVSEM|
-    CLONE_NEWUTS|CLONE_NEWIPC))
+    CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWPIDS))
   goto bad_unshare_out;

  if ((err = unshare_thread(unshare_flags)))
@@ -1642,6 +1642,18 @@ asmlinkage long sys_unshare(unsigned lon
    new_fs)))
   goto bad_unshare_cleanup_semundo;

+ if (new_nsproxy && (unshare_flags & CLONE_NEWPIDS))
+  /*
+   * when we created new pid namespace and failed with something
+   * later we cannot roll back the pid ns creation with simple
+   * put_pid_ns as the task will stay in this namespace holding
+   * it. there are two solutions - pull the task out of this ns
+   * on the error path or push the task into it on the sucess one
+   * I use the second way.
+   */
+  if (move_init_to_ns(current, new_nsproxy))
+   goto bad_move_to_ns;
+
  if (new_fs ||  new_mm || new_fd || new_ulist || new_nsproxy) {

   task_lock(current);
@@ -1676,6 +1688,7 @@ asmlinkage long sys_unshare(unsigned lon
   task_unlock(current);
  }

+bad_move_to_ns:
  if (new_nsproxy)
   put_nsproxy(new_nsproxy);
```

```
--- ./kernel/nsproxy.c.clonepidns 2007-06-15 15:00:32.000000000 +0400
+++ ./kernel/nsproxy.c 2007-06-15 15:14:33.000000000 +0400
@@ -110,6 +110,9 @@ int copy_namespaces(int flags, struct ta

  get_nsproxy(old_ns);

+ if (flags & CLONE_NEWPIDS)
+  return -EINVAL;
+
  if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC)))
   return 0;

@@ -154,7 +157,8 @@ int unshare_nsproxy_namespaces(unsigned
  struct nsproxy *old_ns = current->nsproxy;
  int err = 0;

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

 #ifndef CONFIG_IPC_NS
--- ./kernel/pid.c.clonepidns 2007-06-15 15:13:07.000000000 +0400
+++ ./kernel/pid.c 2007-06-15 15:14:33.000000000 +0400
@@ -28,6 +28,7 @@
 #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)
 static struct hlist_head *pid_hash;
@@ -401,11 +402,102 @@ struct pid *find_ge_pid(int nr, struct p
 }
 EXPORT_SYMBOL_GPL(find_get_pid);

+#ifdef CONFIG_PID_NS
+static struct pid_namespace *create_pid_namespace(void)
+{
+ 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;
+
+ if (pid_ns_prepare_proc(ns))
+  goto out_free_map;
+
+ set_bit(0, ns->pidmap[0].page);
+ atomic_set(&ns->pidmap[0].nr_free, BITS_PER_PAGE - 1);
+
+ kref_init(&ns->kref);
+ ns->last_pid = 0;
+ ns->child_reaper = NULL;
+
+ 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_map:
+ kfree(ns->pidmap[0].page);
+out_free:
+ kfree(ns);
+out:
+ return ERR_PTR(-ENOMEM);
+}
+
+static int alone_in_pgrp(struct task_struct *tsk)
+{
+ int alone = 0;
+ struct pid *pid;
+ struct task_struct *p;
+
+ if (!thread_group_empty(tsk))
+  return 0;
+
+ read_lock(&tasklist_lock);
+ pid = tsk->pids[PIDTYPE_PGID].pid;
+ do_each_pid_task(pid, PIDTYPE_PGID, p) {
+  if (p != tsk)
+   goto out;
+ } while_each_pid_task(pid, PIDTYPE_PGID, p);
+ pid = tsk->pids[PIDTYPE_SID].pid;
+ do_each_pid_task(pid, PIDTYPE_SID, p) {
+  if (p != tsk)
+   goto out;
+ } while_each_pid_task(pid, PIDTYPE_SID, p);
```

```
+ alone = 1;
+out:
+ read_unlock(&tasklist_lock);
+ return alone;
+}
+
+static void destroy_pid_namespace(struct pid_namespace *ns)
+{
+ int i;
+
+ pid_ns_release_proc(ns);
+
+ for (i = 0; i < PIDMAP_ENTRIES; i++)
+  kfree(ns->pidmap[i].page);
+
+ kfree(ns);
+}
+
 struct pid_namespace *copy_pid_ns(int 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(-EBUSY);
+ if (!alone_in_pgrp(current))
+  goto out_put;
+
+ new_ns = create_pid_namespace();
+out_put:
+ put_pid_ns(old_ns);
+out:
+ return new_ns;
 }

 void free_pid_ns(struct kref *kref)
@@ -413,8 +505,64 @@ void free_pid_ns(struct kref *kref)
  struct pid_namespace *ns;

  ns = container_of(kref, struct pid_namespace, kref);
- kfree(ns);
+ destroy_pid_namespace(ns);
+}
```

```
+
+int move_init_to_ns(struct task_struct *tsk, struct nsproxy *nsp)
+{
+ int err;
+ struct pid *pid;
+ struct pid_namespace *ns;
+
+ BUG_ON(tsk != current);
+ ns = nsp->pid_ns;
+
+ pid = task_pid(tsk);
+ err = move_pid_to_ns(pid, ns);
+ if (err < 0)
+  goto out_pid;
+
+ set_task_vpid(tsk, pid_vnr(pid));
+ set_task_vtgid(tsk, pid_vnr(pid));
+
+ pid = task_session(tsk);
+ err = move_pid_to_ns(pid, ns);
+ if (err < 0)
+  goto out_sid;
+
+ set_task_vsession(tsk, pid_vnr(pid));
+
+ pid = task_pgrp(tsk);
+ err = move_pid_to_ns(pid, ns);
+ if (err < 0)
+  goto out_pgid;
+
+ set_task_vpgrp(tsk, pid_vnr(pid));
+
+ ns->child_reaper = tsk;
+ return 0;
+
+out_pgid:
+ del_pid_from_ns(task_session(tsk), ns);
+out_sid:
+ del_pid_from_ns(task_pid(tsk), ns);
+out_pid:
+ return err;
 }
+#else
+struct pid_namespace *copy_pid_ns(int flags, struct pid_namespace *old_ns)
+{
+ if (flags & CLONE_NEWPIDS)
+  old_ns = ERR_PTR(-EINVAL);
+
```

```
+ return old_ns;
+}
+
+int move_init_to_ns(struct task_struct *tsk, struct nsproxy *nsp)
+{
+ BUG();
+}
+#endif

 /*
  * The pid hash table is scaled according to the amount of memory in the
```

Subject: [PATCH 16/28] [FLAT 1/6] Changes in data structures for flat model
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:16:35 GMT
View Forum Message <> Reply to Message

This patch opens the flat model patches.

The flat model idea is that struct pid has two numbers. The first one
(pid->nr) is a global one and is unique in the system. The second one
(pid->vnr) is a virtual pid. It is used on the kernel user boundary only.

I.e. it is shown to user via getpid(), proc, etc, and when the application
passes pid to the kernel to make something with proc, this number is treated
as the virtual one and the pid/task is searched in the namespace caller task
belongs to.

The struct pid must have two numerical values, the pointer to the namespace and
additional hlist_node to hash the pid in two hash-tables.

The virtual ids are stored on struct task_struct and struct signal together with
their global pairs for optimisation.

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

```
 pid.h   |   19 ++++++++++++++++++
 sched.h |   10 +++++++++-
 2 files changed, 28 insertions(+), 1 deletion(-)
```

```
--- ./include/linux/pid.h.flatdatast 2007-06-15 15:08:39.000000000 +0400
+++ ./include/linux/pid.h 2007-06-15 15:22:18.000000000 +0400
@@ -40,12 +40,31 @@ enum pid_type
  * processes.
  */

+/*
+ * flat pid namespaces.
+ * each task hash two ids of each type - the global id and the virtual one
+ * the latter one is used on kernel-user boundary only. i.e. if the kernel
+ * wants to save the task's pid for some time it may store the global one
+ * and the use find_pid()/find_task_by_pid() routines as it was used before.
+ * when an id comes from the usersoace or it must go to user the virtual
+ * id must be used.
+ */
+
 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;
+#ifdef CONFIG_PID_NS_FLAT
+ /*
+  * virtual number, the namespace this pid belongs to and the hlist_node
+  * to find this pid by vnr in hash
+  */
+ int vnr;
+ struct pid_namespace *ns;
+ struct hlist_node vpid_chain;
+#endif
  /* lists of tasks that use this pid */
  struct hlist_head tasks[PIDTYPE_MAX];
  struct rcu_head rcu;
--- ./include/linux/sched.h.flatdatast 2007-06-15 15:14:33.000000000 +0400
+++ ./include/linux/sched.h 2007-06-15 15:19:14.000000000 +0400
@@ -482,7 +482,10 @@ struct signal_struct {
   pid_t session __deprecated;
   pid_t __session;
  };
-
+#ifdef CONFIG_PID_NS_FLAT
+ pid_t vpgrp;
+ pid_t vsession;
+#endif
  /* boolean value for session group leader */
  int leader;
```

```
@@ -944,6 +947,11 @@ struct task_struct {
  unsigned did_exec:1;
  pid_t pid;
  pid_t tgid;
+#ifdef CONFIG_PID_NS_FLAT
+ /* hash the virtual ids as well */
+ pid_t vpid;
+ pid_t vtgid;
+#endif

 #ifdef CONFIG_CC_STACKPROTECTOR
  /* Canary value for the -fstack-protector gcc feature */
```

_____

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

---

Subject: [PATCH 17/28] [FLAT 2/6] Helpers to obtain pid numbers
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:17:36 GMT
View Forum Message <> Reply to Message

The is the implementation of [PREP 2/14] patch for the flat model

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

```
 pid.h   |   27 +++++++++++++++++
 sched.h |  102
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 2 files changed, 129 insertions(+)

--- ./include/linux/pid.h.flatnrs 2007-06-15 15:22:18.000000000 +0400
+++ ./include/linux/pid.h 2007-06-15 15:23:00.000000000 +0400
@@ -171,6 +171,33 @@ static inline int pid_ns_accessible(stru
  return 1;
 }
 #else
+#ifdef CONFIG_PID_NS_FLAT
+static inline pid_t pid_nr(struct pid *pid)
+{
+ pid_t nr = 0;
+ if (pid)
+  nr = pid->nr;
+ return nr;
+}
+
```

```
+static inline pid_t pid_vnr(struct pid *pid)
+{
+ pid_t nr = 0;
+ if (pid)
+  nr = pid->vnr;
+ return nr;
+}
+
+static inline pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
+{
+ return ns == &init_pid_ns ? pid_nr(pid) : pid_vnr(pid);
+}
+
+static inline int pid_ns_accessible(struct pid_namespace *ns, struct pid *pid)
+{
+ return pid->ns == &init_pid_ns || pid->ns == ns;
+}
+#endif
 #endif

 #define do_each_pid_task(pid, type, task)    \
--- ./include/linux/sched.h.flatnrs 2007-06-15 15:19:14.000000000 +0400
+++ ./include/linux/sched.h 2007-06-15 15:23:00.000000000 +0400
@@ -1302,6 +1302,108 @@ static inline pid_t task_ppid_nr_ns(stru
  return rcu_dereference(tsk->real_parent)->tgid;
 }
 #else
+#ifdef CONFIG_PID_NS_FLAT
+static inline pid_t task_pid_nr(struct task_struct *tsk)
+{
+ return tsk->pid;
+}
+
+static inline pid_t task_pid_vnr(struct task_struct *tsk)
+{
+ return tsk->vpid;
+}
+
+static inline pid_t task_pid_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ return (ns == &init_pid_ns ?
+   task_pid_nr(tsk) : task_pid_vnr(tsk));
+}
+
+static inline void set_task_vpid(struct task_struct *tsk, pid_t nr)
+{
+ tsk->vpid = nr;
```

```
+}
+
+
+static inline pid_t task_tgid_nr(struct task_struct *tsk)
+{
+ return tsk->tgid;
+}
+
+static inline pid_t task_tgid_vnr(struct task_struct *tsk)
+{
+ return tsk->vtgid;
+}
+
+static inline pid_t task_tgid_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ return (ns == &init_pid_ns ?
+   task_tgid_nr(tsk) : task_tgid_vnr(tsk));
+}
+
+static inline void set_task_vtgid(struct task_struct *tsk, pid_t nr)
+{
+ tsk->vtgid = nr;
+}
+
+
+static inline pid_t task_pgrp_nr(struct task_struct *tsk)
+{
+ return tsk->signal->pgrp;
+}
+
+static inline pid_t task_pgrp_vnr(struct task_struct *tsk)
+{
+ return tsk->signal->vpgrp;
+}
+
+static inline pid_t task_pgrp_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ return (ns == &init_pid_ns ?
+   task_pgrp_nr(tsk) : task_pgrp_vnr(tsk));
+}
+
+static inline void set_task_vpgrp(struct task_struct *tsk, pid_t nr)
+{
+ tsk->signal->vpgrp = nr;
+}
+
```

```
+
+static inline pid_t task_session_nr(struct task_struct *tsk)
+{
+ return tsk->signal->__session;
+}
+
+static inline pid_t task_session_vnr(struct task_struct *tsk)
+{
+ return tsk->signal->vsession;
+}
+
+static inline pid_t task_session_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ return (ns == &init_pid_ns ?
+   task_session_nr(tsk) : task_session_vnr(tsk));
+}
+
+static inline void set_task_vsession(struct task_struct *tsk, pid_t nr)
+{
+ tsk->signal->vsession = nr;
+}
+
+
+static inline pid_t task_ppid_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ if (tsk->vpid == 1)
+  return 0;
+
+ return rcu_dereference(tsk->real_parent)->vtgid;
+}
+#endif
 #endif

 /**
```


Subject: [PATCH 18/28] [FLAT 3/6] Flush dentries from the namespace's proc tree

Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:18:30 GMT

View Forum Message <> Reply to Message

This is the implementation of [PREP 7/14] for the flat model

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

---

 base.c | 7 +++++++
 1 files changed, 7 insertions(+)

--- ./fs/proc/base.c.flatflush 2007-06-15 15:04:18.000000000 +0400
+++ ./fs/proc/base.c 2007-06-15 15:23:33.000000000 +0400
@@ -2240,6 +2240,13 @@ static inline void proc_flush_task_ns(st
 {
 }
 #else
+#ifdef CONFIG_PID_NS_FLAT
+static inline void proc_flush_task_ns(struct task_struct *tsk, struct pid *pid)
+{
+ if (pid->ns != &init_pid_ns)
+ proc_flush_task_mnt(tsk, pid->ns->proc_mnt);
+}
+#endif
 #endif

 void proc_flush_task(struct task_struct *task, struct pid *pid)

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

---

Subject: [PATCH 19/28] [FLAT 4/6] init_pid initialization
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:19:15 GMT
View Forum Message <> Reply to Message

Initialize numerical for flat model (see [PREP 5/14])

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

---

 init_task.h | 8 ++++++++
 1 files changed, 8 insertions(+)

--- ./include/linux/init_task.h.flatinitpid 2007-06-15 15:01:59.000000000 +0400
+++ ./include/linux/init_task.h 2007-06-15 15:24:00.000000000 +0400
@@ -96,6 +96,14 @@ extern struct group_info init_groups;
  .pid_chain = { .next = NULL, .pprev = NULL }, \

 #else

```
+#ifdef CONFIG_PID_NS_FLAT
+#define INIT_STRUCT_PID_NRS      \
+ .nr  = 0,      \
+ .vnr  = 0,      \
+ .ns  = &init_pid_ns,     \
+ .pid_chain = { .next = NULL, .pprev = NULL },  \
+
+#endif
 #endif

 #define INIT_STRUCT_PID {      \
```

Subject: [PATCH 20/28] [FLAT 5/6] Flat model pid manipulations
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:20:19 GMT
View Forum Message <> Reply to Message

The alloc_pid(), free_pid() and put_pid() implementation for flat
model (see [PREP 10/14]). This model allocates ids from two maps and
hashes the pid in two tables.

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

```
 pid.c |   99
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 files changed, 99 insertions(+)

--- ./kernel/pid.c.flatcore 2007-06-15 15:14:33.000000000 +0400
+++ ./kernel/pid.c 2007-06-15 15:26:23.000000000 +0400
@@ -64,6 +64,12 @@ static inline int mk_pid(struct pid_name
 static struct hlist_head *pid_hash2;
 #define pid_ehashfn(nr, ns) hash_long((unsigned long)nr + (unsigned long)ns, \
     pidhash_shift)
+
+#ifdef CONFIG_PID_NS_FLAT
+/* flat model uses the second hash to find the pids by their virtual nrs */
+#define vpid_hash pid_hash2
+#define vpid_hashfn pid_ehashfn
+#endif
 #endif

 /*
```

```
@@ -233,6 +239,99 @@ struct pid * fastcall find_pid_ns(int nr
  return NULL;
 }
 #else
+#ifdef CONFIG_PID_NS_FLAT
+static inline int alloc_pid_nrs(struct pid *pid)
+{
+ int nr, vnr;
+ struct pid_namespace *ns;
+
+ vnr = nr = alloc_pidmap(&init_pid_ns);
+ if (nr < 0)
+  goto out;
+
+ ns = current->nsproxy->pid_ns;
+ /*
+  * pids in init namespace have both nr and vnr equal
+  * pids in subnamespace hold the namespace
+  */
+ if (ns != &init_pid_ns) {
+  vnr = alloc_pidmap(ns);
+  if (vnr < 0)
+   goto out_vnr;
+
+  get_pid_ns(ns);
+ }
+
+ pid->nr = nr;
+ pid->vnr = vnr;
+ pid->ns = ns;
+ spin_lock_irq(&pidmap_lock);
+ hlist_add_head_rcu(&pid->pid_chain, &pid_hash[pid_hashfn(nr)]);
+ if (ns != &init_pid_ns)
+  hlist_add_head_rcu(&pid->vpid_chain,
+    &vpid_hash[vpid_hashfn(vnr, ns)]);
+ spin_unlock_irq(&pidmap_lock);
+ return 0;
+
+out_vnr:
+ free_pidmap(&init_pid_ns, nr);
+out:
+ return vnr;
+}
+
+static inline void unhash_pid_nrs(struct pid *pid)
+{
+ unsigned long flags;
+
```

```
+ spin_lock_irqsave(&pidmap_lock, flags);
+ hlist_del_rcu(&pid->pid_chain);
+ if (pid->ns != &init_pid_ns)
+  hlist_del_rcu(&pid->vpid_chain);
+ spin_unlock_irqrestore(&pidmap_lock, flags);
+
+ free_pidmap(&init_pid_ns, pid->nr);
+ if (pid->ns != &init_pid_ns)
+  free_pidmap(pid->ns, pid->vnr);
+}
+
+static inline void free_pid_nrs(struct pid *pid)
+{
+ if (pid->ns != &init_pid_ns)
+  put_pid_ns(pid->ns);
+}
+
+static inline struct pid *find_global_pid(int nr)
+{
+ struct pid *pid;
+ struct hlist_node *elem;
+
+ hlist_for_each_entry_rcu(pid, elem,
+   &pid_hash[pid_hashfn(nr)], pid_chain) {
+  if (pid->nr == nr)
+   return pid;
+ }
+ return NULL;
+}
+
+static inline struct pid *find_virtual_pid(int nr, struct pid_namespace *ns)
+{
+ struct pid *pid;
+ struct hlist_node *elem;
+
+ hlist_for_each_entry_rcu(pid, elem,
+   &vpid_hash[vpid_hashfn(nr, ns)], vpid_chain) {
+  if (pid->vnr == nr && pid->ns == ns)
+   return pid;
+ }
+ return NULL;
+}
+
+struct pid * fastcall find_pid_ns(int nr, struct pid_namespace *ns)
+{
+ return (ns == &init_pid_ns ?
+   find_global_pid(nr) : find_virtual_pid(nr, ns));
+}
```

```
+#endif
 #endif

 EXPORT_SYMBOL_GPL(find_pid_ns);
```


Subject: [PATCH 21/28] [FLAT 6/6] Moving the pid into the namespace
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:21:31 GMT
View Forum Message <> Reply to Message

This is the implementation of move_pid_to_ns (see [PATCH 15/28])

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

```
 pid.c |   58 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 files changed, 58 insertions(+)

--- ./kernel/pid.c.flatclone 2007-06-15 15:26:23.000000000 +0400
+++ ./kernel/pid.c 2007-06-15 15:28:22.000000000 +0400
@@ -331,6 +331,48 @@ struct pid * fastcall find_pid_ns(int nr
  return (ns == &init_pid_ns ?
    find_global_pid(nr) : find_virtual_pid(nr, ns));
 }
+
+static inline int move_pid_to_ns(struct pid *pid, struct pid_namespace *ns)
+{
+ int vnr;
+
+ /*
+  * the pid is in this ns already. e.g. this may happen if
+  * the task has equal pid and pgid
+  */
+ if (pid->ns == ns)
+  return 0;
+
+ BUG_ON(pid->ns != &init_pid_ns);
+
+ vnr = alloc_pidmap(ns);
+ if (vnr < 0)
+  return -ENOMEM;
+
+ get_pid_ns(ns);
```

```
+ pid->vnr = vnr;
+ pid->ns = ns;
+ spin_lock_irq(&pidmap_lock);
+ hlist_add_head_rcu(&pid->vpid_chain,
+   &vpid_hash[vpid_hashfn(vnr, ns)]);
+ spin_unlock_irq(&pidmap_lock);
+ return 0;
+}
+
+static inline void del_pid_from_ns(struct pid *pid, struct pid_namespace *ns)
+{
+ if (pid->ns == &init_pid_ns)
+  return;
+
+ spin_lock_irq(&pidmap_lock);
+ hlist_del_rcu(&pid->vpid_chain);
+ spin_unlock_irq(&pidmap_lock);
+
+ free_pidmap(ns, pid->vnr);
+ put_pid_ns(pid->ns);
+ pid->vnr = pid->nr;
+ pid->ns = &init_pid_ns;
+}
 #endif
 #endif

@@ -588,6 +630,15 @@ struct pid_namespace *copy_pid_ns(int fl
  if (!(flags & CLONE_NEWPIDS))
   goto out;

+#ifdef CONFIG_PID_NS_FLAT
+ /*
+  * flat model doen't allow to create the nested namespaces
+  */
+ new_ns = ERR_PTR(-EINVAL);
+ if (old_ns != &init_pid_ns)
+  goto out_put;
+#endif
+
  new_ns = ERR_PTR(-EBUSY);
  if (!alone_in_pgrp(current))
   goto out_put;
@@ -646,6 +697,13 @@ out_pgid:
 out_sid:
  del_pid_from_ns(task_pid(tsk), ns);
 out_pid:
+#ifdef CONFIG_PID_NS_FLAT
+ /*
```

```
+  * this cannot happen if we use flat pid namespaces as we try to
+  * allocate the very first pid from the pidmap with one page in it
+  */
+ BUG();
+#endif
  return err;
 }
 #else
```

Subject: [PATCH 22/28] [MULTI 1/6] Changes in data structures for multilevel model

Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:24:25 GMT

View Forum Message <> Reply to Message

This patch opens the multilevel model patches.

The multilevel model idea is basically the same as for the flat one,
but in this case task may have many virtual pids - one id for each
sub-namespace this task is visible in. The struct pid carries the
list of pid_number-s and two hash tables are used to find this number
by numerical id and by struct pid.

The struct pid doesn't need the numerical ids any longer. Instead it
has a single linked list of struct pid_number-s which are hashed
for quick search and have the numerical id.

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

```
 pid.h |   31 +++++++++++++++++++++++++++++++
 1 files changed, 31 insertions(+)

--- ./include/linux/pid.h.multdatast 2007-06-15 15:23:00.000000000 +0400
+++ ./include/linux/pid.h 2007-06-15 15:32:15.000000000 +0400
@@ -50,9 +50,33 @@ enum pid_type
  * id must be used.
  */

+/*
+ * multilevel pid namespaces
```

```
+ * each task may belong to any number of namespaces and thus struct pid do
+ * not carry the number any longer. instead if this struct pid has a list of
+ * pid_number-s each belonging to one namespace. two hashes are used to find
+ * the number - by the numerical id and by the struct pid this nr belongs to.
+ * this allows for creating namespaces of infinite nesting, but has slight
+ * performance problems.
+ */
+
+struct pid_number
+{
+ int nr;
+ struct pid_namespace *ns;
+ struct pid *pid;
+
+ struct hlist_node pid_chain;
+ struct hlist_node nr_chain;
+ struct pid_number *next;
+};
+
 struct pid
 {
  atomic_t count;
+#ifdef CONFIG_PID_NS_MULTILEVEL
+ struct pid_number *pid_nrs;
+#else
  /* Try to keep pid_chain in the same cacheline as nr for find_pid */
  int nr;
  struct hlist_node pid_chain;
@@ -65,11 +89,18 @@ struct pid
  struct pid_namespace *ns;
  struct hlist_node vpid_chain;
 #endif
+#endif
  /* lists of tasks that use this pid */
  struct hlist_head tasks[PIDTYPE_MAX];
  struct rcu_head rcu;
 };

+#ifdef CONFIG_PID_NS_MULTILEVEL
+/* small helper to iterate over the pid's numbers */
+#define for_each_pid_nr(nr, pid) \
+  for (nr = pid->pid_nrs; nr != NULL; nr = nr->next)
+#endif
+
 extern struct pid init_struct_pid;

 struct pid_link
```

Subject: [PATCH 23/28] [MULTI 2/6] Helpers to obtain pid numbers
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:25:17 GMT
View Forum Message <> Reply to Message

The is the implementation of [PREP 2/14] patch for the multilevel model

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

```
 pid.h   |   30 ++++++++++++++++++++++++++++++
 sched.h |   64
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 2 files changed, 94 insertions(+)

--- ./include/linux/pid.h.multinrs 2007-06-15 15:32:15.000000000 +0400
+++ ./include/linux/pid.h 2007-06-15 15:32:45.000000000 +0400
@@ -229,6 +229,36 @@ static inline int pid_ns_accessible(stru
  return pid->ns == &init_pid_ns || pid->ns == ns;
 }
 #endif
+
+#ifdef CONFIG_PID_NS_MULTILEVEL
+extern struct pid_number *find_nr_by_pid(struct pid *, struct pid_namespace *);
+
+static inline pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
+{
+ struct pid_number *pnr;
+ pid_t nr = 0;
+ if (pid) {
+  pnr = find_nr_by_pid(pid, ns);
+  if (pnr != NULL)
+   nr = pnr->nr;
+ }
+ return nr;
+}
+#define pid_nr(pid) pid_nr_ns(pid, &init_pid_ns)
+#define pid_vnr(pid) pid_nr_ns(pid, current->nsproxy->pid_ns)
+
+static inline int pid_ns_accessible(struct pid_namespace *ns, struct pid *pid)
+{
+ struct pid_number *pnr;
+
```

```
+ /*
+  * the namespace that pid actually lives in is always at the first
+  * pid_number in the list (see alloc_pid_nrs)
+  */
+ pnr = pid->pid_nrs;
+ return pnr->ns == &init_pid_ns || pnr->ns == ns;
+}
+#endif
 #endif

 #define do_each_pid_task(pid, type, task)    \
--- ./include/linux/sched.h.multinrs 2007-06-15 15:23:00.000000000 +0400
+++ ./include/linux/sched.h 2007-06-15 15:32:45.000000000 +0400
@@ -1404,6 +1404,70 @@ static inline pid_t task_ppid_nr_ns(stru
  return rcu_dereference(tsk->real_parent)->vtgid;
 }
 #endif
+
+#ifdef CONFIG_PID_NS_MULTILEVEL
+static inline pid_t task_pid_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ struct pid_number *pnr;
+
+ pnr = find_nr_by_pid(task_pid(tsk), ns);
+ return pnr != NULL ? pnr->nr : 0;
+}
+
+#define task_pid_vnr(t)  task_pid_nr_ns(t, current->nsproxy->pid_ns)
+#define task_pid_nr(t)  task_pid_nr_nr(t, &init_pid_ns)
+#define set_task_vpid(tsk, nr) do { } while (0)
+
+static inline pid_t task_tgid_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ struct pid_number *pnr;
+
+ pnr = find_nr_by_pid(task_tgid(tsk), ns);
+ return pnr != NULL ? pnr->nr : 0;
+}
+
+#define task_tgid_vnr(t) task_tgid_nr_ns(t, current->nsproxy->pid_ns)
+#define task_tgid_nr(t)  task_tgid_nr_ns(t, &init_pid_ns)
+#define set_task_vtgid(tsk, nr) do { } while (0)
+
+static inline pid_t task_pgrp_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
```

```
+ struct pid_number *pnr;
+
+ pnr = find_nr_by_pid(task_pgrp(tsk), ns);
+ return pnr != NULL ? pnr->nr : 0;
+}
+
+#define task_pgrp_vnr(t) task_pgrp_nr_ns(t, current->nsproxy->pid_ns)
+#define task_pgrp_nr(t)  task_pgrp_nr_ns(t, &init_pid_ns)
+#define set_task_vpgrp(tsk, nr) do { } while (0)
+
+static inline pid_t task_session_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ struct pid_number *pnr;
+
+ pnr = find_nr_by_pid(task_session(tsk), ns);
+ return pnr != NULL ? pnr->nr : 0;
+}
+
+#define task_session_vnr(t) task_session_nr_ns(t, current->nsproxy->pid_ns)
+#define task_session_nr(t) task_session_nr_ns(t, &init_pid_ns)
+#define set_task_vsession(tsk, nr) do { } while (0)
+
+static inline pid_t task_ppid_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ struct pid_number *pnr;
+
+ pnr = find_nr_by_pid(task_tgid(rcu_dereference(tsk->real_parent)), ns);
+ return pnr != NULL ? pnr->nr : 0;
+}
+
+#endif
 #endif

 /**
```

Subject: [PATCH 24/28] [MULTI 3/6] Flush dentries from the namespace's proc tree
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:26:13 GMT
View Forum Message <> Reply to Message

This is the implementation of [PREP 7/14] for the multilevel model

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

---

 base.c | 12 ++++++++++++
 1 files changed, 12 insertions(+)

```
--- ./fs/proc/base.c.multiflush 2007-06-15 15:23:33.000000000 +0400
+++ ./fs/proc/base.c 2007-06-15 15:34:17.000000000 +0400
@@ -2247,6 +2247,18 @@ static inline void proc_flush_task_ns(st
   proc_flush_task_mnt(tsk, pid->ns->proc_mnt);
 }
 #endif
+
+#ifdef CONFIG_PID_NS_MULTILEVEL
+static inline void proc_flush_task_ns(struct task_struct *tsk, struct pid *pid)
+{
+ struct pid_number *pnr;
+
+ for_each_pid_nr(pnr, pid)
+  /* init_pid_ns's mnt is global proc_mnt, which is flushed */
+  if (pnr->ns != &init_pid_ns)
+   proc_flush_task_mnt(tsk, pnr->ns->proc_mnt);
+}
+#endif
 #endif

 void proc_flush_task(struct task_struct *task, struct pid *pid)
```

Subject: [PATCH 25/28] [MULTI 4/6] init_pid initialization
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:27:21 GMT
View Forum Message <> Reply to Message

Initialize numericals for multilevel model (see [PREP 5/14])

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

---

 include/linux/init_task.h | 15 +++++++++++++++
 kernel/pid.c | 3 +++
 2 files changed, 18 insertions(+)

```
--- ./include/linux/init_task.h.multiinitpid 2007-06-15 15:24:00.000000000 +0400
+++ ./include/linux/init_task.h 2007-06-15 15:34:43.000000000 +0400
@@ -104,6 +104,21 @@ extern struct group_info init_groups;
  .pid_chain = { .next = NULL, .pprev = NULL },  \

 #endif
+
+#ifdef CONFIG_PID_NS_MULTILEVEL
+#define INIT_PID_NUMBER {      \
+ .nr  = 0,      \
+ .ns  = &init_pid_ns,     \
+ .pid  = &init_struct_pid,    \
+ .pid_chain = { .next = NULL, .pprev = NULL },  \
+ .nr_chain = { .next = NULL, .pprev = NULL },  \
+ .next  = NULL,      \
+}
+
+#define INIT_STRUCT_PID_NRS      \
+ .pid_nrs = &init_pid_number,    \
+
+#endif
 #endif

 #define INIT_STRUCT_PID {      \
--- ./kernel/pid.c.multiinitpid 2007-06-15 15:28:22.000000000 +0400
+++ ./kernel/pid.c 2007-06-15 15:34:43.000000000 +0400
@@ -34,6 +34,9 @@
 static struct hlist_head *pid_hash;
 static int pidhash_shift;
 static struct kmem_cache *pid_cachep;
+#ifdef CONFIG_PID_NS_MULTILEVEL
+static struct pid_number init_pid_number = INIT_PID_NUMBER;
+#endif
 struct pid init_struct_pid = INIT_STRUCT_PID;

 int pid_max = PID_MAX_DEFAULT;
```

Subject: [PATCH 26/28] [MULTI 5/6] Multilevel model pid manipulations
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:28:27 GMT
View Forum Message <> Reply to Message

The alloc_pid(), free_pid() and put_pid() implementation for multilevel
model (see [PREP 10/14]). This model allocates the appropriate number

and hashed them into two tables.

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

```
 pid.c |  158
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 files changed, 158 insertions(+)

--- ./kernel/pid.c.multicore 2007-06-15 15:38:59.000000000 +0400
+++ ./kernel/pid.c 2007-06-15 15:41:30.000000000 +0400
@@ -73,6 +73,22 @@ static struct hlist_head *pid_hash2;
 #define vpid_hash pid_hash2
 #define vpid_hashfn pid_ehashfn
 #endif
+
+#ifdef CONFIG_PID_NS_MULTILEVEL
+/*
+ * multilevel model stores pid numbers in two hashes
+ *  - one hashes them with numerical id and the namespace
+ *  - the other one - with struct pid and the namespace
+ */
+#define pid_phash pid_hash
+#define pid_nhash pid_hash2
+
+#define pid_phashfn pid_ehashfn
+#define pid_nhashfn pid_ehashfn
+
+struct kmem_cache *pid_nr_cachep;
+#endif
+
 #endif

 /*
@@ -377,6 +393,145 @@ static inline void del_pid_from_ns(struc
  pid->ns = &init_pid_ns;
 }
 #endif
+
+#ifdef CONFIG_PID_NS_MULTILEVEL
+static inline int alloc_pid_nr(struct pid *pid, struct pid_namespace *ns)
+{
+ struct pid_number *pnr;
+ int nr;
+
+ pnr = kmem_cache_alloc(pid_nr_cachep, GFP_KERNEL);
+ if (pnr == NULL)
```

```
+  goto out_nr;
+
+ nr = alloc_pidmap(ns);
+ if (nr < 0)
+  goto out_map;
+
+ get_pid_ns(ns);
+ pnr->nr = nr;
+ pnr->ns = ns;
+ pnr->pid = pid;
+ pnr->next = pid->pid_nrs;
+ pid->pid_nrs = pnr;
+ return 0;
+
+out_map:
+ kmem_cache_free(pid_nr_cachep, pnr);
+out_nr:
+ return -ENOMEM;
+}
+
+static inline void free_pid_nr(struct pid_number *pnr)
+{
+ free_pidmap(pnr->ns, pnr->nr);
+ put_pid_ns(pnr->ns);
+ kmem_cache_free(pid_nr_cachep, pnr);
+}
+
+static inline void hash_pid_nr(struct pid_number *pnr)
+{
+ hlist_add_head_rcu(&pnr->nr_chain,
+    &pid_nhash[pid_nhashfn(pnr->nr, pnr->ns)]);
+ hlist_add_head_rcu(&pnr->pid_chain,
+    &pid_phash[pid_phashfn(pnr->pid, pnr->ns)]);
+}
+
+static inline void unhash_pid_nr(struct pid_number *pnr)
+{
+ hlist_del_rcu(&pnr->nr_chain);
+ hlist_del_rcu(&pnr->pid_chain);
+}
+
+static inline void free_pid_nrs(struct pid *pid)
+{
+ struct pid_number *pnr;
+
+ while (pid->pid_nrs != NULL) {
+  pnr = pid->pid_nrs;
+  pid->pid_nrs = pnr->next;
```

```
+  free_pid_nr(pnr);
+ }
+}
+
+static inline int alloc_pid_nrs(struct pid *pid)
+{
+ int err;
+ struct pid *parent_pid;
+ struct pid_number *pnr;
+
+ pid->pid_nrs = NULL;
+
+ parent_pid = task_pid(current);
+ for_each_pid_nr(pnr, parent_pid) {
+  err = alloc_pid_nr(pid, pnr->ns);
+  if (err < 0)
+   goto out;
+ }
+
+ spin_lock_irq(&pidmap_lock);
+ for_each_pid_nr(pnr, pid)
+  hash_pid_nr(pnr);
+ spin_unlock_irq(&pidmap_lock);
+ return 0;
+
+out:
+ free_pid_nrs(pid);
+ return err;
+}
+
+static inline void unhash_pid_nrs(struct pid *pid)
+{
+ unsigned long flags;
+ struct pid_number *pnr;
+
+ spin_lock_irqsave(&pidmap_lock, flags);
+ for_each_pid_nr(pnr, pid)
+  unhash_pid_nr(pnr);
+ spin_unlock_irqrestore(&pidmap_lock, flags);
+}
+
+struct pid_number *find_nr_by_pid(struct pid *pid, struct pid_namespace *ns)
+{
+ struct pid_number *pnr;
+ struct hlist_node *n;
+
+ hlist_for_each_entry_rcu(pnr, n, &pid_phash[pid_phashfn(pid, ns)],
+   pid_chain)
```

```
+  if (pnr->pid == pid && pnr->ns == ns)
+   return pnr;
+
+ return NULL;
+}
+
+EXPORT_SYMBOL_GPL(find_nr_by_pid);
+
+/*
+ * this function finds the struct pid_number by its numerical id
+ * the name is not so beautiful, but this is an internal function
+ */
+
+static struct pid_number *find_nr_by_nr(pid_t nr, struct pid_namespace *ns)
+{
+ struct pid_number *pnr;
+ struct hlist_node *n;
+
+ hlist_for_each_entry_rcu(pnr, n, &pid_nhash[pid_nhashfn(nr, ns)],
+   nr_chain)
+  if (pnr->nr == nr && pnr->ns == ns)
+   return pnr;
+
+ return NULL;
+}
+
+struct pid * fastcall find_pid_ns(int nr, struct pid_namespace *ns)
+{
+ struct pid_number *pnr;
+
+ pnr = find_nr_by_nr(nr, ns);
+ return pnr != NULL ? pnr->pid : NULL;
+}
+#endif
 #endif

 EXPORT_SYMBOL_GPL(find_pid_ns);
@@ -762,4 +917,7 @@ void __init pidmap_init(void)
  atomic_dec(&init_pid_ns.pidmap[0].nr_free);

  pid_cachep = KMEM_CACHE(pid, SLAB_PANIC);
+#ifdef CONFIG_PID_NS_MULTILEVEL
+ pid_nr_cachep = KMEM_CACHE(pid_number, SLAB_PANIC);
+#endif
 }
```

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

Subject: [PATCH 27/28] [MULTI 6/6] Moving the pid into the namespace
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:29:44 GMT
View Forum Message <> Reply to Message

This is the implementation of move_pid_to_ns for multilevel model
(see [PATCH 15/18] and [FLAT 6/6])

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

---

 pid.c | 42 ++
 1 files changed, 42 insertions(+)

```
--- ./kernel/pid.c.multiclone 2007-06-15 15:41:30.000000000 +0400
+++ ./kernel/pid.c 2007-06-15 15:42:36.000000000 +0400
@@ -531,6 +531,48 @@ struct pid * fastcall find_pid_ns(int nr
  pnr = find_nr_by_nr(nr, ns);
  return pnr != NULL ? pnr->pid : NULL;
 }
+
+static inline int move_pid_to_ns(struct pid *pid, struct pid_namespace *ns)
+{
+ int err;
+
+ /*
+  * the pid is in this ns already. e.g. this may happen if
+  * the task has equal pid and pgid
+  */
+ if (find_nr_by_pid(pid, ns))
+  return 0;
+
+ err = alloc_pid_nr(pid, ns);
+ if (err < 0)
+  return err;
+
+ spin_lock_irq(&pidmap_lock);
+ hash_pid_nr(pid->pid_nrs);
+ spin_unlock_irq(&pidmap_lock);
+ return 0;
+}
+
+static inline void del_pid_from_ns(struct pid *pid, struct pid_namespace *ns)
+{
+ struct pid_number *pnr, *prev;
```

```
+
+ for_each_pid_nr(prev, pid) {
+  pnr = prev->next;
+  if (pnr != NULL && pnr->ns == ns)
+   goto found;
+ }
+ return;
+
+found:
+ prev->next = pnr->next;
+
+ spin_lock_irq(&pidmap_lock);
+ unhash_pid_nr(pnr);
+ spin_unlock_irq(&pidmap_lock);
+
+ free_pid_nr(pnr);
+}
 #endif
 #endif
```

Subject: [PATCH 28/28] CONFIG_PID_NS kconfig option
Posted by Pavel Emelianov on Fri, 15 Jun 2007 16:31:18 GMT
View Forum Message <> Reply to Message

There's CONFIG_PID_NS option to turn on the pid virtualzation
and the "choise" for the model type depending on it.

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

```
 Kconfig |   30 ++++++++++++++++++++++++++++++
 1 files changed, 30 insertions(+)

diff --git a/init/Kconfig b/init/Kconfig
index 2a46e35..6b2cbcf 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -127,6 +127,36 @@ config SWAP_PREFETCH
    Workstations and multiuser workstation servers will most likely want
    to say Y.
```

```
+config PID_NS
+ bool "Pid namespaces"
+ depends on EXPERIMENTAL
+ default n
+ help
+   Enable pid namespaces support. When on task is allowed to unshare
+   its pid namespace from parent and become its init. After this task
+   all its children will see only the tasks from this namespace.
+   However tasks from parent namespace see all the tasks in the system.
+   Ony one level of nesting is alowed. Tasks cannot leave the namespace.
+
+choice
+ prompt "Select pid namespace model"
+ default PID_NS_FLAT
+ depends on PID_NS
+ help
+   This option selects a pid namespace model
+
+config PID_NS_FLAT
+ bool "Flat pid namespace"
+ help
+   Enable only one level of pid namespaces.
+
+config PID_NS_MULTILEVEL
+ bool "Multilevel pid namespaces"
+ help
+   Enable creating pid namespaces of infinite nesting.
+
+endchoice
+
 config SYSVIPC
  bool "System V IPC"
  ---help---
```


Subject: Re: [PATCH 16/28] [FLAT 1/6] Changes in data structures for flat model
Posted by Sukadev Bhattiprolu on Tue, 19 Jun 2007 05:14:54 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@openvz.org] wrote:
| This patch opens the flat model patches.
|
| The flat model idea is that struct pid has two numbers. The first one

| (pid->nr) is a global one and is unique in the system. The second one
| (pid->vnr) is a virtual pid. It is used on the kernel user boundary only.

This approach duplicates 5 integers and 2 pointers per process for every
process in the system. While this may not be expensive for processes that
actually use multiple namespaces, doesn't it waste memory if majority of
processes exist only in one namespace ?

|
| I.e. it is shown to user via getpid(), proc, etc, and when the application
| passes pid to the kernel to make something with proc, this number is treated
| as the virtual one and the pid/task is searched in the namespace caller task
| belongs to.
|
|
|
| The struct pid must have two numerical values, the pointer to the namespace and
| additional hlist_node to hash the pid in two hash-tables.
|
| The virtual ids are stored on struct task_struct and struct signal together with
| their global pairs for optimisation.
|
| Signed-off-by: Pavel Emelianov <xemul@openvz.org>
|
| ---
|
| pid.h | 19 ++++++++++++++++++++
| sched.h | 10 +++++++++-
| 2 files changed, 28 insertions(+), 1 deletion(-)
|
| --- ./include/linux/pid.h.flatdatast 2007-06-15 15:08:39.000000000 +0400
| +++ ./include/linux/pid.h 2007-06-15 15:22:18.000000000 +0400
| @@ -40,12 +40,31 @@ enum pid_type
| * processes.
| */
|
| +/*
| + * flat pid namespaces.
| + * each task hash two ids of each type - the global id and the virtual one
| + * the latter one is used on kernel-user boundary only. i.e. if the kernel
| + * wants to save the task's pid for some time it may store the global one
| + * and the use find_pid()/find_task_by_pid() routines as it was used before.
| + * when an id comes from the usersoace or it must go to user the virtual
| + * id must be used.
| + */
| +
| 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;
| +#ifdef CONFIG_PID_NS_FLAT
| + /*
| +  * virtual number, the namespace this pid belongs to and the hlist_node
| +  * to find this pid by vnr in hash
| +  */
| + int vnr;
| + struct pid_namespace *ns;
| + struct hlist_node vpid_chain;
| +#endif
|   /* lists of tasks that use this pid */
|   struct hlist_head tasks[PIDTYPE_MAX];
|   struct rcu_head rcu;
| --- ./include/linux/sched.h.flatdatast 2007-06-15 15:14:33.000000000 +0400
| +++ ./include/linux/sched.h 2007-06-15 15:19:14.000000000 +0400
| @@ -482,7 +482,10 @@ struct signal_struct {
|    pid_t session __deprecated;
|    pid_t __session;
|   };
| -
| +#ifdef CONFIG_PID_NS_FLAT
| + pid_t vpgrp;
| + pid_t vsession;
| +#endif
|   /* boolean value for session group leader */
|   int leader;
|
| @@ -944,6 +947,11 @@ struct task_struct {
|   unsigned did_exec:1;
|   pid_t pid;
|   pid_t tgid;
| +#ifdef CONFIG_PID_NS_FLAT
| + /* hash the virtual ids as well */
| + pid_t vpid;
| + pid_t vtgid;
| +#endif
|
|  #ifdef CONFIG_CC_STACKPROTECTOR
|   /* Canary value for the -fstack-protector gcc feature */
```

Subject: Re: [PATCH 22/28] [MULTI 1/6] Changes in data structures for multilevel model
Posted by Sukadev Bhattiprolu on Tue, 19 Jun 2007 06:32:50 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@openvz.org] wrote:
| This patch opens the multilevel model patches.
|
| The multilevel model idea is basically the same as for the flat one,
| but in this case task may have many virtual pids - one id for each
| sub-namespace this task is visible in. The struct pid carries the
| list of pid_number-s and two hash tables are used to find this number
| by numerical id and by struct pid.
|
|
|
| The struct pid doesn't need the numerical ids any longer. Instead it
| has a single linked list of struct pid_number-s which are hashed
| for quick search and have the numerical id.
|
| Signed-off-by: Pavel Emelianov <xemul@openvz.org>
|
| ---
|
| pid.h | 31 +++++++++++++++++++++++++++++++
| 1 files changed, 31 insertions(+)
|
| --- ./include/linux/pid.h.multdatast 2007-06-15 15:23:00.000000000 +0400
| +++ ./include/linux/pid.h 2007-06-15 15:32:15.000000000 +0400
| @@ -50,9 +50,33 @@ enum pid_type
| * id must be used.
| */
|
| +/*
| + * multilevel pid namespaces
| + * each task may belong to any number of namespaces and thus struct pid do
| + * not carry the number any longer. instead if this struct pid has a list of
| + * pid_number-s each belonging to one namespace. two hashes are used to find
| + * the number - by the numerical id and by the struct pid this nr belongs to.
| + * this allows for creating namespaces of infinite nesting, but has slight
| + * performance problems.
| + */
| +
| +struct pid_number
| +{
| + int nr;
| + struct pid_namespace *ns;
| + struct pid *pid;
| +

```
| + struct hlist_node pid_chain;
| + struct hlist_node nr_chain;
| + struct pid_number *next;
```

As you probably noticed, we had a similar linked list until recently.
But since we use only clone() to create a new pid namespace, we figured
we could use an array of 'struct pid_number' elements. That may perform
slightly better since all 'pid_number elements' are co-located.

We obviously need a list like this if we unshare (rather than clone())
pid namespace.

I have a few questions - not that I see any problems yet - just for my
understanding (they may be addressed in other patches, but am still
reviewing them).

- Can one process unshare() its namespace, create a few children,
  and unshare its namespace again ?

- If so, will that same process be the reaper for multiple pid
  namespaces ?

- Will we terminate all those namespaces if the reaper is terminated ?

```
| +};
| +
|  struct pid
|  {
|   atomic_t count;
| +#ifdef CONFIG_PID_NS_MULTILEVEL
| + struct pid_number *pid_nrs;
| +#else
|   /* Try to keep pid_chain in the same cacheline as nr for find_pid */
|   int nr;
|   struct hlist_node pid_chain;
| @@ -65,11 +89,18 @@ struct pid
|   struct pid_namespace *ns;
|   struct hlist_node vpid_chain;
|  #endif
| +#endif
|   /* lists of tasks that use this pid */
|   struct hlist_head tasks[PIDTYPE_MAX];
|   struct rcu_head rcu;
|  };
|
| +#ifdef CONFIG_PID_NS_MULTILEVEL
| +/* small helper to iterate over the pid's numbers */
```

| +#define for_each_pid_nr(nr, pid) \
| + for (nr = pid->pid_nrs; nr != NULL; nr = nr->next)
| +#endif
| +
| extern struct pid init_struct_pid;
|
| struct pid_link

_____

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

---

Subject: Re: [PATCH 16/28] [FLAT 1/6] Changes in data structures for flat model
Posted by Sukadev Bhattiprolu on Tue, 19 Jun 2007 06:52:35 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 patch opens the flat model patches.
| > |
| > | The flat model idea is that struct pid has two numbers. The first one
| > | (pid->nr) is a global one and is unique in the system. The second one
| > | (pid->vnr) is a virtual pid. It is used on the kernel user boundary only.
| >
| > This approach duplicates 5 integers and 2 pointers per process for every
| > process in the system. While this may not be expensive for processes that
| > actually use multiple namespaces, doesn't it waste memory if majority of
| > processes exist only in one namespace ?
|
| task_struct alignment allows for it. so does the alignment of signal structure.
| and please note that this comes with appropriate ifdefs around. the only problem
| is with struct pid, but we're virtualizing it after all!

Hmm. I don't understand the last part "we are virtualizing 'struct pid'".
Even so, with the FLAT model, every process will still have two
pid_t values, two hash-chain links etc - no ?

|
| moreover - two integers and a pointer to the namespace is the minimal set of
| fields for pid that is visible from two namespaces...

I ignored the pid_namespace pointer. But even a process that exists only
in init_pid_ns would have the extra fields right ?

_____

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

Subject: Re: [PATCH 16/28] [FLAT 1/6] Changes in data structures for flat model
Posted by Pavel Emelianov on Tue, 19 Jun 2007 07:18:17 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Pavel Emelianov [xemul@openvz.org] wrote:
> | This patch opens the flat model patches.
> |
> | The flat model idea is that struct pid has two numbers. The first one
> | (pid->nr) is a global one and is unique in the system. The second one
> | (pid->vnr) is a virtual pid. It is used on the kernel user boundary only.
>
> This approach duplicates 5 integers and 2 pointers per process for every
> process in the system. While this may not be expensive for processes that
> actually use multiple namespaces, doesn't it waste memory if majority of
> processes exist only in one namespace ?

task_struct alignment allows for it. so does the alignment of signal structure.
and please note that this comes with appropriate ifdefs around. the only problem
is with struct pid, but we're virtualizing it after all!

moreover - two integers and a pointer to the namespace is the minimal set of
fields for pid that is visible from two namespaces...

> |
> | I.e. it is shown to user via getpid(), proc, etc, and when the application
> | passes pid to the kernel to make something with proc, this number is treated
> | as the virtual one and the pid/task is searched in the namespace caller task
> | belongs to.
> |
> |
> |
> | The struct pid must have two numerical values, the pointer to the namespace and
> | additional hlist_node to hash the pid in two hash-tables.
> |
> | The virtual ids are stored on struct task_struct and struct signal together with
> | their global pairs for optimisation.
> |
> | Signed-off-by: Pavel Emelianov <xemul@openvz.org>
> |
> | ---
> |
> | pid.h | 19 ++++++++++++++++++
> | sched.h | 10 +++++++++-
> | 2 files changed, 28 insertions(+), 1 deletion(-)

```
> |
> | --- ./include/linux/pid.h.flatdatast 2007-06-15 15:08:39.000000000 +0400
> | +++ ./include/linux/pid.h 2007-06-15 15:22:18.000000000 +0400
> | @@ -40,12 +40,31 @@ enum pid_type
> |   * processes.
> |   */
> |
> | +/*
> | + * flat pid namespaces.
> | + * each task hash two ids of each type - the global id and the virtual one
> | + * the latter one is used on kernel-user boundary only. i.e. if the kernel
> | + * wants to save the task's pid for some time it may store the global one
> | + * and the use find_pid()/find_task_by_pid() routines as it was used before.
> | + * when an id comes from the usersoace or it must go to user the virtual
> | + * id must be used.
> | + */
> | +
> |  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;
> | +#ifdef CONFIG_PID_NS_FLAT
> | + /*
> | +  * virtual number, the namespace this pid belongs to and the hlist_node
> | +  * to find this pid by vnr in hash
> | +  */
> | + int vnr;
> | + struct pid_namespace *ns;
> | + struct hlist_node vpid_chain;
> | +#endif
> |   /* lists of tasks that use this pid */
> |   struct hlist_head tasks[PIDTYPE_MAX];
> |   struct rcu_head rcu;
> | --- ./include/linux/sched.h.flatdatast 2007-06-15 15:14:33.000000000 +0400
> | +++ ./include/linux/sched.h 2007-06-15 15:19:14.000000000 +0400
> | @@ -482,7 +482,10 @@ struct signal_struct {
> |    pid_t session __deprecated;
> |    pid_t __session;
> |   };
> | -
> | +#ifdef CONFIG_PID_NS_FLAT
> | + pid_t vpgrp;
> | + pid_t vsession;
> | +#endif
> |   /* boolean value for session group leader */
> |   int leader;
```

> |
> | @@ -944,6 +947,11 @@ struct task_struct {
> | unsigned did_exec:1;
> | pid_t pid;
> | pid_t tgid;
> | +#ifdef CONFIG_PID_NS_FLAT
> | + /* hash the virtual ids as well */
> | + pid_t vpid;
> | + pid_t vtgid;
> | +#endif
> |
> | #ifdef CONFIG_CC_STACKPROTECTOR
> | /* Canary value for the -fstack-protector gcc feature */
>

_____

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

---

Subject: Re: Re: [PATCH 22/28] [MULTI 1/6] Changes in data structures for multilevel model
Posted by Pavel Emelianov on Tue, 19 Jun 2007 07:49:05 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Pavel Emelianov [xemul@openvz.org] wrote:
> | This patch opens the multilevel model patches.
> |
> | The multilevel model idea is basically the same as for the flat one,
> | but in this case task may have many virtual pids - one id for each
> | sub-namespace this task is visible in. The struct pid carries the
> | list of pid_number-s and two hash tables are used to find this number
> | by numerical id and by struct pid.
> |
> |
> |
> | The struct pid doesn't need the numerical ids any longer. Instead it
> | has a single linked list of struct pid_number-s which are hashed
> | for quick search and have the numerical id.
> |
> | Signed-off-by: Pavel Emelianov <xemul@openvz.org>
> |
> | ---
> |
> | pid.h | 31 ++++++++++++++++++++++++++++++++
> | 1 files changed, 31 insertions(+)

> |
> | --- ./include/linux/pid.h.multdatast 2007-06-15 15:23:00.000000000 +0400
> | +++ ./include/linux/pid.h 2007-06-15 15:32:15.000000000 +0400
> | @@ -50,9 +50,33 @@ enum pid_type
> | * id must be used.
> | */
> |
> | +/*
> | + * multilevel pid namespaces
> | + * each task may belong to any number of namespaces and thus struct pid do
> | + * not carry the number any longer. instead if this struct pid has a list of
> | + * pid_number-s each belonging to one namespace. two hashes are used to find
> | + * the number - by the numerical id and by the struct pid this nr belongs to.
> | + * this allows for creating namespaces of infinite nesting, but has slight
> | + * performance problems.
> | + */
> | +
> | +struct pid_number
> | +{
> | + int nr;
> | + struct pid_namespace *ns;
> | + struct pid *pid;
> | +
> | + struct hlist_node pid_chain;
> | + struct hlist_node nr_chain;
> | + struct pid_number *next;
>
> As you probably noticed, we had a similar linked list until recently.
> But since we use only clone() to create a new pid namespace, we figured
> we could use an array of 'struct pid_number' elements. That may perform
> slightly better since all 'pid_number elements' are co-located.

Yes, I know it. This ability is a good reason to clone the namespace
via clone()...

> We obviously need a list like this if we unshare (rather than clone())
> pid namespace.
>
> I have a few questions - not that I see any problems yet - just for my
> understanding (they may be addressed in other patches, but am still
> reviewing them).
>
> - Can one process unshare() its namespace, create a few children,
>   and unshare its namespace again ?

Yes, it can.

> - If so, will that same process be the reaper for multiple pid

> namespaces ?

It will. The process that created the namespace will become its reaper.
But I think this is wrong... Reaper should be such for the only namespace.

> - Will we terminate all those namespaces if the reaper is terminated ?

As you will see I do not terminate the namespace on reaper's death.

But it looks like you have caught a BUG in my patches - when a task
is a reaper for multiple namespaces and when he exits the namespaces
will point to the exited task as a reaper :(I will fix it.

>
> | +};
> | +
> | struct pid
> | {
> | atomic_t count;
> | +#ifdef CONFIG_PID_NS_MULTILEVEL
> | + struct pid_number *pid_nrs;
> | +#else
> | /* Try to keep pid_chain in the same cacheline as nr for find_pid */
> | int nr;
> | struct hlist_node pid_chain;
> | @@ -65,11 +89,18 @@ struct pid
> | struct pid_namespace *ns;
> | struct hlist_node vpid_chain;
> | #endif
> | +#endif
> | /* lists of tasks that use this pid */
> | struct hlist_head tasks[PIDTYPE_MAX];
> | struct rcu_head rcu;
> | };
> |
> | +#ifdef CONFIG_PID_NS_MULTILEVEL
> | +/* small helper to iterate over the pid's numbers */
> | +#define for_each_pid_nr(nr, pid) \
> | + for (nr = pid->pid_nrs; nr != NULL; nr = nr->next)
> | +#endif
> | +
> | extern struct pid init_struct_pid;
> |
> | struct pid_link
> _____
> 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: Re: [PATCH 16/28] [FLAT 1/6] Changes in data structures for flat model
Posted by Pavel Emelianov on Tue, 19 Jun 2007 07:52:29 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:
> | > | This patch opens the flat model patches.
> | > |
> | > | The flat model idea is that struct pid has two numbers. The first one
> | > | (pid->nr) is a global one and is unique in the system. The second one
> | > | (pid->vnr) is a virtual pid. It is used on the kernel user boundary only.
> | >
> | > This approach duplicates 5 integers and 2 pointers per process for every
> | > process in the system. While this may not be expensive for processes that
> | > actually use multiple namespaces, doesn't it waste memory if majority of
> | > processes exist only in one namespace ?
> |
> | task_struct alignment allows for it. so does the alignment of signal structure.
> | and please note that this comes with appropriate ifdefs around. the only problem
> | is with struct pid, but we're virtualizing it after all!
>
> Hmm. I don't understand the last part "we are virtualizing 'struct pid'".
> Even so, with the FLAT model, every process will still have two
> pid_t values, two hash-chain links etc - no ?

I mean that since we're adding some extra functionality to the kernel
this is OK to extend the data structures. Moreover we have these changes
under appropriate ifdefs.

> |
> | moreover - two integers and a pointer to the namespace is the minimal set of
> | fields for pid that is visible from two namespaces...
>
> I ignored the pid_namespace pointer. But even a process that exists only
> in init_pid_ns would have the extra fields right ?

It will. But this is OK.

> _____
> 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 1/28] [PREP 1/14] Round up the API
Posted by Cedric Le Goater on Tue, 19 Jun 2007 10:00:04 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>

I think we all agreed on this one. Can you send it to lkml@ as a
candidate for -mm ?

Thanks,

C.

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

---

Subject: Re: [PATCH 1/28] [PREP 1/14] Round up the API
Posted by Pavel Emelianov on Tue, 19 Jun 2007 11:01:02 GMT

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>
>
> I think we all agreed on this one. Can you send it to lkml@ as a
> candidate for -mm ?

I already did that actually. But Andrew didn't answer so I
decided that he considers this patch to be the part of the
pid namespaces.

I will resend it this week separately if get no other
response from him.

> Thanks,
>
> C.
>

Thanks,
Pavel

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

Subject: Re: [PATCH 16/28] [FLAT 1/6] Changes in data structures for flat model
Posted by Dave Hansen on Tue, 19 Jun 2007 16:27:28 GMT

On Tue, 2007-06-19 at 11:18 +0400, Pavel Emelianov wrote:
> sukadev@us.ibm.com wrote:
> > Pavel Emelianov [xemul@openvz.org] wrote:
> > | This patch opens the flat model patches.
> > |
> > | The flat model idea is that struct pid has two numbers. The first one
> > | (pid->nr) is a global one and is unique in the system. The second one
> > | (pid->vnr) is a virtual pid. It is used on the kernel user boundary only.
> > >
> > This approach duplicates 5 integers and 2 pointers per process for every
> > process in the system. While this may not be expensive for processes that
> > actually use multiple namespaces, doesn't it waste memory if majority of
> > processes exist only in one namespace ?
> >
> task_struct alignment allows for it. so does the alignment of signal structure.
> and please note that this comes with appropriate ifdefs around. the only problem
> is with struct pid, but we're virtualizing it after all!

The #ifdefs do not help at all in the real world. Distributions will
ship one and only one kernel configuration, save for things like SMP.
This means that, no matter what, we're going to have the config option
on.

So, for distribution customers, consider the #ifdef turned on all the
time.

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

---

Subject: Re: [PATCH 16/28] [FLAT 1/6] Changes in data structures for flat model
Posted by ebiederm on Tue, 19 Jun 2007 19:11:13 GMT
View Forum Message <> Reply to Message

Pavel Emelianov <xemul@openvz.org> writes:

> sukadev@us.ibm.com wrote:
>> Pavel Emelianov [xemul@openvz.org] wrote:
>> | This patch opens the flat model patches.
>> |
>> | The flat model idea is that struct pid has two numbers. The first one
>> | (pid->nr) is a global one and is unique in the system. The second one
>> | (pid->vnr) is a virtual pid. It is used on the kernel user boundary only.
>> >
>> This approach duplicates 5 integers and 2 pointers per process for every
>> process in the system. While this may not be expensive for processes that

>> actually use multiple namespaces, doesn't it waste memory if majority of
>> processes exist only in one namespace ?
>
> task_struct alignment allows for it. so does the alignment of signal structure.
> and please note that this comes with appropriate ifdefs around. the only problem
> is with struct pid, but we're virtualizing it after all!
>
> moreover - two integers and a pointer to the namespace is the minimal set of
> fields for pid that is visible from two namespaces...

>> | --- ./include/linux/sched.h.flatdatast 2007-06-15 15:14:33.000000000 +0400
>> | +++ ./include/linux/sched.h 2007-06-15 15:19:14.000000000 +0400
>> | @@ -482,7 +482,10 @@ struct signal_struct {
>> | pid_t session __deprecated;
>> | pid_t __session;
>> | };
>> | -
>> | +#ifdef CONFIG_PID_NS_FLAT
>> | + pid_t vpgrp;
>> | + pid_t vsession;
>> | +#endif
>> | /* boolean value for session group leader */
>> | int leader;
>> |
>> | @@ -944,6 +947,11 @@ struct task_struct {
>> | unsigned did_exec:1;
>> | pid_t pid;
>> | pid_t tgid;
>> | +#ifdef CONFIG_PID_NS_FLAT
>> | + /* hash the virtual ids as well */
>> | + pid_t vpid;
>> | + pid_t vtgid;
>> | +#endif

Adding vpgrp, vsession, vpid, and vtgid is wrong.

A case can probably be made for caching the common case (users view),
but we already have fields for that.

For a global view we must use struct pid *, otherwise we are just asking
for trouble.

Eric

Subject: Re: [PATCH 0/28] Pid namespaces (two models)
Posted by akpm on Tue, 19 Jun 2007 23:00:09 GMT
View Forum Message <> Reply to Message

On Fri, 15 Jun 2007 19:55:43 +0400
Pavel Emelianov <xemul@openvz.org> wrote:

> Long ago Sukadev and I sent two approaches for pid namespaces - the
> hierarchical model in which namespaces are nested into each other,
> and the flat model, where pids have only two values and creation of
> level 3 namespace is prohibited.
>
> After that I showed that multilevel model introduces a noticeable
> overhead of approximately 1-2% to kernel standard operations like
> fork() and getpid(). At the same time flat model showed no performance
> hit on these tests.
>
> Nevertheless multilevel model is worth living.
>
> This set introduces booth models each under its config option. The
> set is logically splitted into the following parts:

Making this configurable sounds like a very bad idea to me, from the
maintainablility/testability/understandability POV.

We should just make up our minds and do it one way, do it right?

I assume that means hierarchical.

> The following tests were run:
> [1] nptl perf test
> [2] getpid() speed
> [3] ltp (not for speed, but for kernel API checks)
>
> The testing results summary:
> * Flat model provides zero overhead in init namespace for all the
> tests and less than 7% in the namespace for nptl test only.
> * Multilevel model provides up to 2% overhead in init namespace and
> more than 10% for nptl test in the level 2 namespace.
>

So that means we take a 3% hit in these operations when PID_NS_MULTILEVEL
is enabled but the system isn't using containers at all?

If so, I'm surprised that the cost is this high. This should be the first
thing we should optimise and I bet there's some quicky way of doing it.

_____
Containers mailing list

Subject: Re: [PATCH 16/28] [FLAT 1/6] Changes in data structures for flat model
Posted by Pavel Emelianov on Wed, 20 Jun 2007 10:53:14 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Pavel Emelianov <xemul@openvz.org> writes:
>

[snip]

>>> | --- ./include/linux/sched.h.flatdatast 2007-06-15 15:14:33.000000000 +0400
>>> | +++ ./include/linux/sched.h 2007-06-15 15:19:14.000000000 +0400
>>> | @@ -482,7 +482,10 @@ struct signal_struct {
>>> | pid_t session __deprecated;
>>> | pid_t __session;
>>> | };
>>> | -
>>> | +#ifdef CONFIG_PID_NS_FLAT
>>> | + pid_t vpgrp;
>>> | + pid_t vsession;
>>> | +#endif
>>> | /* boolean value for session group leader */
>>> | int leader;
>>> |
>>> | @@ -944,6 +947,11 @@ struct task_struct {
>>> | unsigned did_exec:1;
>>> | pid_t pid;
>>> | pid_t tgid;
>>> | +#ifdef CONFIG_PID_NS_FLAT
>>> | + /* hash the virtual ids as well */
>>> | + pid_t vpid;
>>> | + pid_t vtgid;
>>> | +#endif
>
> Adding vpgrp, vsession, vpid, and vtgid is wrong.
>
> A case can probably be made for caching the common case (users view),
> but we already have fields for that.
>
> For a global view we must use struct pid *, otherwise we are just asking
> for trouble.

Nope. If we have global unique numerical pid we're not asking for
trouble. We're just making kernel work like it always did. Virtual

pid makes sense *only* when interacting with user level.

Making task->pid virtual is asking for trouble.

> Eric

Pavel

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

---

Subject: Re: [PATCH 16/28] [FLAT 1/6] Changes in data structures for flat model
Posted by ebiederm on Wed, 20 Jun 2007 17:24:23 GMT
View Forum Message <> Reply to Message

Pavel Emelianov <xemul@openvz.org> writes:

> Eric W. Biederman wrote:
>> Pavel Emelianov <xemul@openvz.org> writes:
>>
>
> [snip]
>
>>>> | --- ./include/linux/sched.h.flatdatast 2007-06-15 15:14:33.000000000 +0400
>>>> | +++ ./include/linux/sched.h 2007-06-15 15:19:14.000000000 +0400
>>>> | @@ -482,7 +482,10 @@ struct signal_struct {
>>>> | pid_t session __deprecated;
>>>> | pid_t __session;
>>>> | };
>>>> | -
>>>> | +#ifdef CONFIG_PID_NS_FLAT
>>>> | + pid_t vpgrp;
>>>> | + pid_t vsession;
>>>> | +#endif
>>>> | /* boolean value for session group leader */
>>>> | int leader;
>>>> |
>>>> | @@ -944,6 +947,11 @@ struct task_struct {
>>>> | unsigned did_exec:1;
>>>> | pid_t pid;
>>>> | pid_t tgid;
>>>> | +#ifdef CONFIG_PID_NS_FLAT
>>>> | + /* hash the virtual ids as well */
>>>> | + pid_t vpid;
>>>> | + pid_t vtgid;
>>>> | +#endif

>>
>> Adding vpgrp, vsession, vpid, and vtgid is wrong.
>>
>> A case can probably be made for caching the common case (users view),
>> but we already have fields for that.
>>
>> For a global view we must use struct pid *, otherwise we are just asking
>> for trouble.
>
> Nope. If we have global unique numerical pid we're not asking for
> trouble. We're just making kernel work like it always did. Virtual
> pid makes sense *only* when interacting with user level.
>
> Making task->pid virtual is asking for trouble.

I'm not talking about making it virtual. Virtualization is the wrong concept. We aren't virtualizing something that isn't already virtualized. We are changing the implementation of an abstraction.

I'm talking about keeping task->pid as what the user space sees. Full stop.

The only point of even retaining task->pid or any of the other numeric pid identifiers on struct task_struct, struct signal_struct in data structures outside of struct pid is as an optimization. To make it clearer what we are doing we may want to rename the fields in task_struct and signal_struct but in no sense can I see use needing to cache anything except the common case which is what user space sees.

None of the implementations in the kernel needs a global numeric identifier for processes. In all cases a pointer to a struct pid is just as good from a uniqueness perspective. And all of the heavy lifting has already been done, to change the in kernel users to use struct pid pointers. There are only a few remaining cases that need to be touched now and those cases are the cases where semantically things matter.

There is very much a human interface issue with needing global numeric process ids (as single dense namespace for things is easier for people to wrap their heads around). However we have global numeric ids that live in struct pid, and show up in the init_pid_namespace. So that is not an issue.

Given this conversation I think it is time to investigate if the
optimization of reading pid values from the task struct instead of
struct pid is measurable.

Eric

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

---

Subject: Re: [PATCH 16/28] [FLAT 1/6] Changes in data structures for flat model
Posted by ebiederm on Wed, 20 Jun 2007 17:36:57 GMT
View Forum Message <> Reply to Message

Pavel Emelianov <xemul@openvz.org> writes:

> Eric W. Biederman wrote:
>> Pavel Emelianov <xemul@openvz.org> writes:
>>
>
> [snip]
>
>>>> | --- ./include/linux/sched.h.flatdatast 2007-06-15 15:14:33.000000000 +0400
>>>> | +++ ./include/linux/sched.h 2007-06-15 15:19:14.000000000 +0400
>>>> | @@ -482,7 +482,10 @@ struct signal_struct {
>>>> | pid_t session __deprecated;
>>>> | pid_t __session;
>>>> | };
>>>> | -
>>>> | +#ifdef CONFIG_PID_NS_FLAT
>>>> | + pid_t vpgrp;
>>>> | + pid_t vsession;
>>>> | +#endif
>>>> | /* boolean value for session group leader */
>>>> | int leader;
>>>> |
>>>> | @@ -944,6 +947,11 @@ struct task_struct {
>>>> | unsigned did_exec:1;
>>>> | pid_t pid;
>>>> | pid_t tgid;
>>>> | +#ifdef CONFIG_PID_NS_FLAT
>>>> | + /* hash the virtual ids as well */
>>>> | + pid_t vpid;
>>>> | + pid_t vtgid;
>>>> | +#endif
>>

>> Adding vpgrp, vsession, vpid, and vtgid is wrong.
>>
>> A case can probably be made for caching the common case (users view),
>> but we already have fields for that.
>>
>> For a global view we must use struct pid *, otherwise we are just asking
>> for trouble.
>
> Nope. If we have global unique numerical pid we're not asking for
> trouble. We're just making kernel work like it always did. Virtual
> pid makes sense *only* when interacting with user level.
>
> Making task->pid virtual is asking for trouble.

I'm not talking about making it virtual. Virtualization is the wrong concept. We aren't virtualizing something that isn't already virtualized. We are changing the implementation of an abstraction.

I'm talking about keeping task->pid as what the user space sees. Full stop.

The only point of even retaining task->pid or any of the other numeric pid identifiers on struct task_struct, struct signal_struct in data structures outside of struct pid is as an optimization. To make it clearer what we are doing we may want to rename the fields in task_struct and signal_struct but in no sense can I see use needing to cache anything except the common case which is what user space sees.

None of the implementations in the kernel needs a global numeric identifier for processes. In all cases a pointer to a struct pid is just as good from a uniqueness perspective. And all of the heavy lifting has already been done, to change the in kernel users to use struct pid pointers. There are only a few remaining cases that need to be touched now and those cases are the cases where semantically things matter.

There is very much a human interface issue with needing global numeric process ids (as single dense namespace for things is easier for people to wrap their heads around). However we have global numeric ids that live in struct pid, and show up in the init_pid_namespace. So that is not an issue.

Given this conversation I think it is time to investigate if the
optimization of reading pid values from the task struct instead of
struct pid is measurable.

Eric

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

---

Subject: Re: [PATCH 13/28] [PREP 13/14] Miscellaneous preparations in pid namespaces
Posted by Sukadev Bhattiprolu on Wed, 20 Jun 2007 21:10:13 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@openvz.org] wrote:
| The most important one is moving exit_task_namespaces behind exit_notify
| in do_exit() to make it possible to see the task's pid namespace to
| properly notify the parent.

Hmm. I think we tried this once a few months ago and got a crash in nfsd
See http://lkml.org/lkml/2007/1/17/75

```
[<c01f6115>] lockd_down+0x125/0x190
[<c01d26bd>] nfs_free_server+0x6d/0xd0
[<c01d8e9c>] nfs_kill_super+0xc/0x20
[<c0161c5d>] deactivate_super+0x7d/0xa0
[<c0175e0e>] release_mounts+0x6e/0x80
[<c0175e86>] __put_mnt_ns+0x66/0x80
[<c0132b3e>] free_nsproxy+0x5e/0x60
   // exit_task_namespaces() after returning from exit_notify()
[<c011f021>] do_exit+0x791/0x810
[<c011f0c6>] do_group_exit+0x26/0x70
[<c0103142>] sysenter_past_esp+0x5f/0x85
```

exit_notify() sets current->sighand to NULL and I think lockd_down() called
from exit_task_namespaces/__put_mnt_ns() was accesssing current->sighand.

Do your other patches in this set tweak something to prevent it ?

Thats one of the reasons we had to remove pid_ns from nsproxy and use
the pid_ns from pid->upid_list[i]->pid_ns.

Suka

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

Subject: Re: [PATCH 13/28] [PREP 13/14] Miscellaneous preparations in pid namespaces
Posted by Pavel Emelianov on Fri, 22 Jun 2007 01:10:04 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Pavel Emelianov [xemul@openvz.org] wrote:
> | The most important one is moving exit_task_namespaces behind exit_notify
> | in do_exit() to make it possible to see the task's pid namespace to
> | properly notify the parent.
>
> Hmm. I think we tried this once a few months ago and got a crash in nfsd
> See http://lkml.org/lkml/2007/1/17/75
>
> [<c01f6115>] lockd_down+0x125/0x190
> [<c01d26bd>] nfs_free_server+0x6d/0xd0
> [<c01d8e9c>] nfs_kill_super+0xc/0x20
> [<c0161c5d>] deactivate_super+0x7d/0xa0
> [<c0175e0e>] release_mounts+0x6e/0x80
> [<c0175e86>] __put_mnt_ns+0x66/0x80
> [<c0132b3e>] free_nsproxy+0x5e/0x60
> // exit_task_namespaces() after returning from exit_notify()
> [<c011f021>] do_exit+0x791/0x810
> [<c011f0c6>] do_group_exit+0x26/0x70
> [<c0103142>] sysenter_past_esp+0x5f/0x85
>
> exit_notify() sets current->sighand to NULL and I think lockd_down() called
> from exit_task_namespaces/__put_mnt_ns() was accesssing current->sighand.

If sighand is set to NULL and then accessed then how is this related to pid namespace?

> Do your other patches in this set tweak something to prevent it ?

I think no. I'll check it for my current patches.

> Thats one of the reasons we had to remove pid_ns from nsproxy and use
> the pid_ns from pid->upid_list[i]->pid_ns.
>
> Suka
>

Subject: Re: Re: [PATCH 13/28] [PREP 13/14] Miscellaneous preparations in pid namespaces
Posted by Pavel Emelianov on Fri, 22 Jun 2007 01:27:53 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Pavel Emelianov [xemul@openvz.org] wrote:
> | The most important one is moving exit_task_namespaces behind exit_notify
> | in do_exit() to make it possible to see the task's pid namespace to
> | properly notify the parent.
>
> Hmm. I think we tried this once a few months ago and got a crash in nfsd
> See http://lkml.org/lkml/2007/1/17/75
>
> [<c01f6115>] lockd_down+0x125/0x190
> [<c01d26bd>] nfs_free_server+0x6d/0xd0
> [<c01d8e9c>] nfs_kill_super+0xc/0x20
> [<c0161c5d>] deactivate_super+0x7d/0xa0
> [<c0175e0e>] release_mounts+0x6e/0x80
> [<c0175e86>] __put_mnt_ns+0x66/0x80
> [<c0132b3e>] free_nsproxy+0x5e/0x60
> // exit_task_namespaces() after returning from exit_notify()
> [<c011f021>] do_exit+0x791/0x810
> [<c011f0c6>] do_group_exit+0x26/0x70
> [<c0103142>] sysenter_past_esp+0x5f/0x85
>
> exit_notify() sets current->sighand to NULL and I think lockd_down() called
> from exit_task_namespaces/__put_mnt_ns() was accesssing current->sighand.

OK. I've got it. That's easy - no need to give up the nsproxy->pid_ns.
I will show it in the next patches. Hope to send them today.

Thank

> Do your other patches in this set tweak something to prevent it ?
>
> Thats one of the reasons we had to remove pid_ns from nsproxy and use
> the pid_ns from pid->upid_list[i]->pid_ns.
>
> Suka
> _____
> 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 13/28] [PREP 13/14] Miscellaneous preparations in pid namespaces
Posted by Sukadev Bhattiprolu on Fri, 22 Jun 2007 16:32:56 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@openvz.org] wrote:
| sukadev@us.ibm.com wrote:
| > Pavel Emelianov [xemul@openvz.org] wrote:
| > | The most important one is moving exit_task_namespaces behind exit_notify
| > | in do_exit() to make it possible to see the task's pid namespace to
| > | properly notify the parent.
| >
| > Hmm. I think we tried this once a few months ago and got a crash in nfsd
| > See http://lkml.org/lkml/2007/1/17/75
| >
| > [<c01f6115>] lockd_down+0x125/0x190
| > [<c01d26bd>] nfs_free_server+0x6d/0xd0
| > [<c01d8e9c>] nfs_kill_super+0xc/0x20
| > [<c0161c5d>] deactivate_super+0x7d/0xa0
| > [<c0175e0e>] release_mounts+0x6e/0x80
| > [<c0175e86>] __put_mnt_ns+0x66/0x80
| > [<c0132b3e>] free_nsproxy+0x5e/0x60
| > // exit_task_namespaces() after returning from exit_notify()
| > [<c011f021>] do_exit+0x791/0x810
| > [<c011f0c6>] do_group_exit+0x26/0x70
| > [<c0103142>] sysenter_past_esp+0x5f/0x85
| >
| > exit_notify() sets current->sighand to NULL and I think lockd_down() called
| > from exit_task_namespaces/__put_mnt_ns() was accesssing current->sighand.
|
| If sighand is set to NULL and then accessed then how is this related to pid namespace?

Switching the order of exit_notify() and exit_task_namespaces() is what
caused the problem when we did it before.

If you exit_task_namespaces() before exit_notify() as the mainline code
does, you won't see this bc nfsd would have freed its super by then.

|
| > Do your other patches in this set tweak something to prevent it ?
|
| I think no. I'll check it for my current patches.

Buried in that thread was a test case to repro the problem. Maybe that
will help.

|
| > Thats one of the reasons we had to remove pid_ns from nsproxy and use
| > the pid_ns from pid->upid_list[i]->pid_ns.

| >
| > Suka
| >

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

---

Subject: Re: [PATCH 0/28] Pid namespaces (two models)
Posted by Herbert Poetzl on Wed, 27 Jun 2007 04:01:14 GMT
View Forum Message <> Reply to Message

On Tue, Jun 19, 2007 at 04:00:09PM -0700, Andrew Morton wrote:
> On Fri, 15 Jun 2007 19:55:43 +0400
> Pavel Emelianov <xemul@openvz.org> wrote:
>
> > Long ago Sukadev and I sent two approaches for pid namespaces - the
> > hierarchical model in which namespaces are nested into each other,
> > and the flat model, where pids have only two values and creation of
> > level 3 namespace is prohibited.
> >
> > After that I showed that multilevel model introduces a noticeable
> > overhead of approximately 1-2% to kernel standard operations like
> > fork() and getpid(). At the same time flat model showed no performance
> > hit on these tests.
> >
> > Nevertheless multilevel model is worth living.
> >
> > This set introduces booth models each under its config option. The
> > set is logically splitted into the following parts:
>
> Making this configurable sounds like a very bad idea to me, from the
> maintainablility/testability/understandability POV.
>
> We should just make up our minds and do it one way, do it right?
>
> I assume that means hierarchical.
>
> > The following tests were run:
> > [1] nptl perf test
> > [2] getpid() speed
> > [3] ltp (not for speed, but for kernel API checks)
> >
> > The testing results summary:
> > * Flat model provides zero overhead in init namespace for all the
> > tests and less than 7% in the namespace for nptl test only.

why do we see 7% overhead in nptl tests?
any idea what actually causes that?

TIA,
Herbert

> > * Multilevel model provides up to 2% overhead in init namespace and
> > more than 10% for nptl test in the level 2 namespace.
> >
>
> So that means we take a 3% hit in these operations when PID_NS_MULTILEVEL
> is enabled but the system isn't using containers at all?
>
> If so, I'm surprised that the cost is this high. This should be the first
> thing we should optimise and I bet there's some quicky way of doing it.
>
> _____
> 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