Subject: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by Pavel Emelianov on Thu, 24 May 2007 12:32:44 GMT
View Forum Message <> Reply to Message

That's how OpenVZ sees the pid namespaces.

The main idea is that kernel keeps operating with tasks pid
as it did before, but each task obtains one more pid for each
pid type - the virtual pid. When putting the pid to user or
getting the pid from it kernel operates with the virtual ones.

E.g. virtual pid is returned from getpid(), virtual pgid -
from getpgid() and so on. Getting virtual pid from user is
performed in setpgid(), setsid() and kill() mainly and in some
other places.

As far as the namespace are concerned I propose the following
scheme. The namespace can be created from unshare syscall only.
This makes fork() code look easier. Of course task must be
prepared to have its pids changed. When task creates a new
namespace it becomes its init and sees the tasks from it only.
Tasks from init namespace see all the tasks.

One relevant thing left behind is shrinking both proc's entries
on task death. The reason I didn't do that is the following: this
does not guarantee that the pid will be put (and thus still may
hold the namespace), but makes the patch more complicated. So if
this set will turns out to be interesting I will implement this
thing as well.

The patches are for 2.6.22-rc1-mm1 tree.

Thanks,
Pavel

Subject: [PATCH 1/13] Round up the API
Posted by Pavel Emelianov on Thu, 24 May 2007 12:35:20 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.

I've sent this before, but Andrew didn't include it, so I resend it
as the part of this set.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>
Acked-by: Serge E. Hallyn <serue@us.ibm.com>

```
diff --git a/arch/mips/kernel/irixelf.c b/arch/mips/kernel/irixelf.c
index 403d96f..10ba0a5 100644
--- a/arch/mips/kernel/irixelf.c
+++ b/arch/mips/kernel/irixelf.c
@@ -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
diff --git a/arch/mips/kernel/irixsig.c b/arch/mips/kernel/irixsig.c
index 6980deb..210503e 100644
--- a/arch/mips/kernel/irixsig.c
+++ b/arch/mips/kernel/irixsig.c
@@ -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;
diff --git a/arch/mips/kernel/sysirix.c b/arch/mips/kernel/sysirix.c
index 93a1484..23c3e82 100644
--- a/arch/mips/kernel/sysirix.c
+++ b/arch/mips/kernel/sysirix.c
@@ -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;
diff --git a/arch/sparc64/solaris/misc.c b/arch/sparc64/solaris/misc.c
index 3b67de7..c86cb30 100644
--- a/arch/sparc64/solaris/misc.c
+++ b/arch/sparc64/solaris/misc.c
@@ -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 */
   {
diff --git a/drivers/char/tty_io.c b/drivers/char/tty_io.c
index 4251904..260a1f3 100644
--- a/drivers/char/tty_io.c
+++ b/drivers/char/tty_io.c
@@ -3486,7 +3486,7 @@ 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);
@@ -3496,7 +3496,7 @@ 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;
diff --git a/fs/autofs/inode.c b/fs/autofs/inode.c
index e7204d7..45f5992 100644
--- a/fs/autofs/inode.c
+++ b/fs/autofs/inode.c
@@ -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;

diff --git a/fs/autofs/root.c b/fs/autofs/root.c
index c148953..592f640 100644
--- a/fs/autofs/root.c
+++ b/fs/autofs/root.c
@@ -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)
diff --git a/fs/autofs4/autofs_i.h b/fs/autofs4/autofs_i.h
index d85f42f..2d4ae40 100644
--- a/fs/autofs4/autofs_i.h
```

```
+++ b/fs/autofs4/autofs_i.h
@@ -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? */
diff --git a/fs/autofs4/inode.c b/fs/autofs4/inode.c
index 692364e..32a39b0 100644
--- a/fs/autofs4/inode.c
+++ b/fs/autofs4/inode.c
@@ -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;
diff --git a/fs/autofs4/root.c b/fs/autofs4/root.c
index 2d4c8a3..c766ff8 100644
--- a/fs/autofs4/root.c
+++ b/fs/autofs4/root.c
@@ -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)
diff --git a/fs/binfmt_elf.c b/fs/binfmt_elf.c
index fa8ea33..7893feb 100644
--- a/fs/binfmt_elf.c
+++ b/fs/binfmt_elf.c
@@ -1327,8 +1327,8 @@ 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
@@ -1373,8 +1373,8 @@ 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;
diff --git a/fs/binfmt_elf_fdpic.c b/fs/binfmt_elf_fdpic.c
index 9d62fba..9bb9ff1 100644
--- a/fs/binfmt_elf_fdpic.c
+++ b/fs/binfmt_elf_fdpic.c
@@ -1334,8 +1334,8 @@ 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
@@ -1383,8 +1383,8 @@ 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;
diff --git a/fs/coda/upcall.c b/fs/coda/upcall.c
index a5b5e63..3c35721 100644
--- a/fs/coda/upcall.c
+++ b/fs/coda/upcall.c
@@ -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;
diff --git a/fs/proc/array.c b/fs/proc/array.c
index e798e11..aef7b7b 100644
--- a/fs/proc/array.c
+++ b/fs/proc/array.c
@@ -381,8 +381,8 @@ static int do_task_stat(struct task_stru
    stime = cputime_add(stime, 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);
diff --git a/include/linux/sched.h b/include/linux/sched.h
index 335dfc5..d4de6d8 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1092,24 +1092,19 @@ 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)
+static inline pid_t task_session_nr(struct task_struct *tsk)
 {
- return sig->__session;
+ return tsk->signal->__session;
 }

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

 static inline struct pid *task_pid(struct task_struct *task)
diff --git a/kernel/exit.c b/kernel/exit.c
index c6d14b8..43ce25b 100644
--- a/kernel/exit.c
+++ b/kernel/exit.c
@@ -308,12 +308,12 @@ 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));
@@ -1050,10 +1050,10 @@ 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;
  }

diff --git a/kernel/fork.c b/kernel/fork.c
index 48928b1..d7207a1 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -1249,8 +1249,8 @@ 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));

diff --git a/kernel/signal.c b/kernel/signal.c
index 3c09ee4..75c5d77 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -506,7 +506,7 @@ 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))
diff --git a/kernel/sys.c b/kernel/sys.c
index e0e2da9..8aefd5e 100644
--- a/kernel/sys.c
+++ b/kernel/sys.c
@@ -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;
```

Subject: [PATCH 2/13] Small preparations for namespaces
Posted by Pavel Emelianov on Thu, 24 May 2007 12:37:31 GMT
View Forum Message <> Reply to Message

This includes #ifdefs in get/put_pid_ns and rewriting
the child_reaper() function to the more logical view.

This doesn't fit logically into any other patch so
I decided to make it separate.

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

```
diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
index 169c6c2..7af7191 100644
--- a/include/linux/pid_namespace.h
+++ b/include/linux/pid_namespace.h
@@ -26,7 +26,9 @@ 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);
@@ -34,12 +36,15 @@ 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 */
```

Subject: [PATCH 3/13] Introduciton of config option and clone flag
Posted by Pavel Emelianov on Thu, 24 May 2007 12:42:04 GMT
View Forum Message <> Reply to Message

The config option is CONFIG_PID_NS. The flag is CLONE_NEWPIDS.

As I have already said - cloning of pid namespace from fork()
is not allowed - use unshare for this.

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

```
diff --git a/init/Kconfig b/init/Kconfig
index 2a46e35..59e4625 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -127,6 +127,16 @@ config SWAP_PREFETCH
    Workstations and multiuser workstation servers will most likely want
    to say Y.

+config PID_NS
+ bool "Pid namespaces"
+ 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.
+
 config SYSVIPC
  bool "System V IPC"
  ---help---
diff --git a/include/linux/sched.h b/include/linux/sched.h
index d4de6d8..7743a11 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -26,6 +26,7 @@
```

```
 #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
diff --git a/kernel/fork.c b/kernel/fork.c
index d7207a1..3ab517c 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -1606,7 +1612,7 @@ 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)))
diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index 1bc4b55..9bcc047 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -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
@@ -166,6 +170,10 @@ int unshare_nsproxy_namespaces(unsigned
  if (unshare_flags & CLONE_NEWUTS)
   return -EINVAL;
 #endif
```

```
+#ifndef CONFIG_PID_NS
+ if (unshare_flags & CLONE_NEWPIDS)
+  return -EINVAL;
+#endif

  if (!capable(CAP_SYS_ADMIN))
   return -EPERM;
```

Subject: [PATCH 4/13] Introduce the vpid fields and helpers for getting them
Posted by Pavel Emelianov on Thu, 24 May 2007 12:44:07 GMT

View Forum Message <> Reply to Message

Since we want to export virtual pid to userspace and this is
optional (CONFIG_PID_NS) we need helpers for getting the values
of vpid/vtgid/etc depending on the config and the appropriate
members on structs.

A note about the struct pid. As will be seen later pid may now
be stored in two hashes - pid_hash and the vpid_hash. The latter
is used to find the struct pid by pid get from the userspace.

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

```
diff --git a/include/linux/pid.h b/include/linux/pid.h
index 1e0e4e3..3a30f8a 100644
--- a/include/linux/pid.h
+++ b/include/linux/pid.h
@@ -46,6 +46,11 @@ struct pid
  /* 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
+ int vnr;
+ struct hlist_node vpid_chain;
+ struct pid_namespace *ns;
+#endif
  /* lists of tasks that use this pid */
  struct hlist_head tasks[PIDTYPE_MAX];
  struct rcu_head rcu;
@@ -106,6 +114,20 @@ static inline pid_t pid_nr(struct pid *p
  return nr;
 }

+#ifdef CONFIG_PID_NS
+static inline pid_t pid_vnr(struct pid *pid)
```

```
+{
+ pid_t nr = 0;
+ if (pid)
+  nr = pid->vnr;
+ return nr;
+}
+#else
+#define pid_vnr(pid) pid_nr(pid)
+#endif
+
+#define pid_nr_ns(pid, ns) (ns == &init_pid_ns ? pid_nr(pid) : pid_vnr(pid))
+
 #define do_each_pid_task(pid, type, task)    \
  do {        \
   struct hlist_node *pos___;    \
diff --git a/include/linux/sched.h b/include/linux/sched.h
index d4de6d8..7743a11 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -456,7 +457,10 @@ struct signal_struct {
   pid_t session __deprecated;
   pid_t __session;
  };
-
+#ifdef CONFIG_PID_NS
+ pid_t vpgrp;
+ pid_t vsession;
+#endif
  /* boolean value for session group leader */
  int leader;

@@ -887,6 +891,10 @@ struct task_struct {
  unsigned did_exec:1;
  pid_t pid;
  pid_t tgid;
+#ifdef CONFIG_PID_NS
+ pid_t vpid;
+ pid_t vtgid;
+#endif

 #ifdef CONFIG_CC_STACKPROTECTOR
  /* Canary value for the -fstack-protector gcc feature */
@@ -1127,6 +1135,128 @@ static inline struct pid *task_session(s
  return task->group_leader->pids[PIDTYPE_SID].pid;
 }

+#ifdef CONFIG_PID_NS
+static inline pid_t task_pid_vnr(struct task_struct *tsk)
```

```
+{
+ return tsk->vpid;
+}
+
+static inline void set_task_vpid(struct task_struct *tsk, pid_t nr)
+{
+ tsk->vpid = nr;
+}
+
+static inline pid_t task_tgid_vnr(struct task_struct *tsk)
+{
+ return tsk->vtgid;
+}
+
+static inline void set_task_vtgid(struct task_struct *tsk, pid_t nr)
+{
+ tsk->vtgid = nr;
+}
+
+static inline pid_t task_session_vnr(struct task_struct *tsk)
+{
+ return tsk->signal->vsession;
+}
+
+static inline void set_task_vsession(struct task_struct *tsk, pid_t nr)
+{
+ tsk->signal->vsession = nr;
+}
+
+static inline pid_t task_pgrp_vnr(struct task_struct *tsk)
+{
+ return tsk->signal->vpgrp;
+}
+
+static inline void set_task_vpgrp(struct task_struct *tsk, pid_t nr)
+{
+ tsk->signal->vpgrp = nr;
+}
+
+extern struct pid_namespace init_pid_ns;
+
+static inline pid_t task_ppid_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ if (ns == &init_pid_ns)
+  return rcu_dereference(tsk->real_parent)->tgid;
+
+ if (tsk->vpid == 1)
+  return 0;
+
+ return rcu_dereference(tsk->real_parent)->vtgid;
+}
+#else
+static inline pid_t task_pid_vnr(struct task_struct *tsk)
+{
+ return tsk->pid;
+}
+
+static inline void set_task_vpid(struct task_struct *tsk, pid_t nr)
+{
+}
+
+static inline pid_t task_tgid_vnr(struct task_struct *tsk)
+{
+ return tsk->tgid;
+}
+
+static inline void set_task_vtgid(struct task_struct *tsk, pid_t nr)
+{
+}
+
+static inline pid_t task_session_vnr(struct task_struct *tsk)
+{
+ return task_session_nr(tsk);
+}
+
+static inline void set_task_vsession(struct task_struct *tsk, pid_t nr)
+{
+}
+
+static inline pid_t task_pgrp_vnr(struct task_struct *tsk)
+{
+ return task_pgrp_nr(tsk);
+}
+
+static inline void set_task_vpgrp(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;
+}
+#endif
+
+#define __task_pid_nr_ns(tsk, ns) \
+ (ns == &init_pid_ns ? tsk->pid : task_pid_vnr(tsk))
+
+#define task_pid_nr_ns(tsk)  \
+ __task_pid_nr_ns(tsk, current->nsproxy->pid_ns)
+
+#define __task_tgid_nr_ns(tsk, ns) \
+ (ns == &init_pid_ns ? tsk->tgid : task_tgid_vnr(tsk))
+
+#define task_tgid_nr_ns(tsk)  \
+ __task_tgid_nr_ns(tsk, current->nsproxy->pid_ns)
+
+#define __task_pgrp_nr_ns(tsk, ns) \
+ (ns == &init_pid_ns ? task_pgrp_nr(tsk) : task_pgrp_vnr(tsk))
+
+#define task_pgrp_nr_ns(tsk)  \
+ __task_pgrp_nr_ns(tsk, current->nsproxy->pid_ns)
+
+#define __task_session_nr_ns(tsk, ns) \
+ (ns == &init_pid_ns ? task_session_nr(tsk) : task_session_vnr(tsk))
+
+#define task_session_nr_ns(tsk)  \
+ __task_session_nr_ns(tsk, current->nsproxy->pid_ns)
+
 /**
  * pid_alive - check that a task structure is not stale
  * @p: Task structure to be checked.
```

Subject: [PATCH 5/13] Expand the pid/task seeking functions set
Posted by Pavel Emelianov on Thu, 24 May 2007 12:46:49 GMT

View Forum Message <> Reply to Message

We need the extended set of functions for searching
tasks and pids - search in global namespace, in local
namespace (current belongs to) and in arbitrary namespace
(used in proc).

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

```
diff --git a/include/linux/sched.h b/include/linux/sched.h
index d4de6d8..7743a11 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1298,8 +1428,16 @@ 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;
+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. */
diff --git a/include/linux/pid.h b/include/linux/pid.h
index 1e0e4e3..3a30f8a 100644
--- a/include/linux/pid.h
+++ b/include/linux/pid.h
@@ -83,17 +88,20 @@ 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;
 /*
  * look up a PID in the hash table. Must be called with the tasklist_lock
  * or rcu_read_lock() held.
  */
 extern struct pid *FASTCALL(find_pid(int nr));
+extern struct pid *FASTCALL(__find_vpid(int nr, struct pid_namespace *ns));
+#define find_vpid(pid) __find_vpid(pid, current->nsproxy->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));
diff --git a/kernel/pid.c b/kernel/pid.c
index eb66bd2..1815af4 100644
--- a/kernel/pid.c
+++ b/kernel/pid.c
@@ -249,6 +289,27 @@ struct pid * fastcall find_pid(int nr)
```

```
 }
 EXPORT_SYMBOL_GPL(find_pid);

+struct pid * fastcall __find_vpid(int nr, struct pid_namespace *ns)
+{
+#ifdef CONFIG_PID_NS
+ struct hlist_node *elem;
+ struct pid *pid;
+#endif
+
+ if (ns == &init_pid_ns)
+  return find_pid(nr);
+
+#ifdef CONFIG_PID_NS
+ hlist_for_each_entry_rcu(pid, elem,
+   &vpid_hash[vpid_hashfn(nr, ns)], vpid_chain) {
+  if (pid->vnr == nr && pid->ns == ns)
+   return pid;
+ }
+#endif
+ return NULL;
+}
+EXPORT_SYMBOL_GPL(__find_vpid);
+
 /*
  * attach_pid() must be called with the tasklist_lock write-held.
  */
@@ -307,12 +368,13 @@ 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_vpid(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 +401,7 @@ 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;
@@ -350,15 +412,15 @@ 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_vpid(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 6/13] Pid allocation/freeing procedures
Posted by Pavel Emelianov on Thu, 24 May 2007 12:49:47 GMT

View Forum Message <> Reply to Message

This patch make alloc_pid() and free_pid() aware of the
namespaces. When a pid is created not in init namespace
it gets into two hashes and holds the pointer to the
namespace itself.

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

```
diff --git a/kernel/pid.c b/kernel/pid.c
index eb66bd2..1815af4 100644
--- a/kernel/pid.c
+++ b/kernel/pid.c
@@ -54,6 +54,12 @@ 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
```

```
+static struct hlist_head *vpid_hash;
+#define vpid_hashfn(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
@@ -197,9 +203,19 @@ fastcall void free_pid(struct pid *pid)

  spin_lock_irqsave(&pidmap_lock, flags);
  hlist_del_rcu(&pid->pid_chain);
+#ifdef CONFIG_PID_NS
+ if (pid->ns != &init_pid_ns)
+  hlist_del_rcu(&pid->vpid_chain);
+#endif
  spin_unlock_irqrestore(&pidmap_lock, flags);

  free_pidmap(&init_pid_ns, pid->nr);
+#ifdef CONFIG_PID_NS
+ if (pid->ns != &init_pid_ns) {
+  free_pidmap(pid->ns, pid->vnr);
+  put_pid_ns(pid->ns);
+ }
+#endif
  call_rcu(&pid->rcu, delayed_put_pid);
 }

@@ -207,28 +223,52 @@ struct pid *alloc_pid(void)
 {
  struct pid *pid;
  enum pid_type type;
- int nr = -1;
+ int nr, vnr;
+ struct pid_namespace *ns;

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

- nr = alloc_pidmap(current->nsproxy->pid_ns);
+ vnr = nr = alloc_pidmap(&init_pid_ns);
  if (nr < 0)
   goto out_free;

+ ns = current->nsproxy->pid_ns;
+#ifdef CONFIG_PID_NS
+ if (ns != &init_pid_ns) {
```

```
+  vnr = alloc_pidmap(ns);
+  if (vnr < 0)
+   goto out_free_map;
+
+  get_pid_ns(ns);
+ }
+#endif
  atomic_set(&pid->count, 1);
  pid->nr = nr;
+#ifdef CONFIG_PID_NS
+ pid->vnr = vnr;
+ pid->ns = ns;
+#endif
  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)]);
+#ifdef CONFIG_PID_NS
+ if (ns != &init_pid_ns)
+  hlist_add_head_rcu(&pid->vpid_chain,
+    &vpid_hash[vpid_hashfn(vnr, ns)]);
+#endif
  spin_unlock_irq(&pidmap_lock);

 out:
  return pid;

+#ifdef CONFIG_PID_NS
+out_free_map:
+ free_pidmap(&init_pid_ns, nr);
+#endif
 out_free:
  kmem_cache_free(pid_cachep, pid);
  pid = NULL;
@@ -397,12 +607,17 @@ 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
+ vpid_hash = pid_hash + (pidhash_size / 2);
+#endif
 }

 void __init pidmap_init(void)
```

Subject: [PATCH 7/13] Set virtual pids for a newly cloned task
Posted by Pavel Emelianov on Thu, 24 May 2007 12:53:23 GMT
View Forum Message <> Reply to Message

When new task is created it must have its virtual pids set.
When task belongs to init namespace the pids are equal to
global ones so it is safe to get vpid from any task.

This is the place where we export pids to use space and there
will be a patch for these cases, but this case is specal and
thus goes separately.

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

```
diff --git a/kernel/fork.c b/kernel/fork.c
index d7207a1..3ab517c 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -1026,6 +1026,7 @@ 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;
@@ -1101,8 +1102,11 @@ 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;
@@ -1251,6 +1255,8 @@ 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));

@@ -1270,7 +1276,7 @@ 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;
@@ -1372,7 +1378,7 @@ 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)
```

Subject: [PATCH 8/13] The namespace cloning
Posted by Pavel Emelianov on Thu, 24 May 2007 12:57:58 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>

```
diff --git a/include/linux/pid.h b/include/linux/pid.h
index 1e0e4e3..3a30f8a 100644
--- a/include/linux/pid.h
+++ b/include/linux/pid.h
@@ -106,6 +114,8 @@ static inline pid_t pid_nr(struct pid *p

 #define pid_nr_ns(pid, ns) (ns == &init_pid_ns ? pid_nr(pid) : pid_vnr(pid))

+void move_init_to_ns(struct task_struct *tsk);
+
 #define do_each_pid_task(pid, type, task)    \
  do {        \
   struct hlist_node *pos___;    \
diff --git a/kernel/fork.c b/kernel/fork.c
index d7207a1..3ab517c 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -1659,8 +1665,11 @@ asmlinkage long sys_unshare(unsigned lon
   task_unlock(current);
  }

- if (new_nsproxy)
+ if (new_nsproxy) {
+  if (unshare_flags & CLONE_NEWPIDS)
+   move_init_to_ns(current);
   put_nsproxy(new_nsproxy);
+ }

 bad_unshare_cleanup_semundo:
 bad_unshare_cleanup_fd:
diff --git a/kernel/pid.c b/kernel/pid.c
index eb66bd2..1815af4 100644
--- a/kernel/pid.c
+++ b/kernel/pid.c
@@ -350,11 +412,99 @@ struct pid *find_get_pid(pid_t nr)
 }
 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;
+
+ 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:
+ 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;
+
+ 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(-EINVAL);
+ if (old_ns != &init_pid_ns)
+  goto out_put;
+
+ 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)
@@ -377,9 +527,69 @@ 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);
 }

+static inline int move_pid_to_ns(struct pid *pid, struct pid_namespace *ns)
+{
+ int vnr;
+
+ 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;
+}
+
+void move_init_to_ns(struct task_struct *tsk)
+{
+ struct pid *pid;
+ struct pid_namespace *ns;
+
+ BUG_ON(tsk != current);
+ ns = tsk->nsproxy->pid_ns;
+
+ pid = tsk->pids[PIDTYPE_PID].pid;
+ BUG_ON(pid->ns != &init_pid_ns);
+ if (move_pid_to_ns(pid, ns))
+  BUG();
+ set_task_vpid(tsk, pid->vnr);
+ set_task_vtgid(tsk, pid->vnr);
+
+ pid = tsk->pids[PIDTYPE_SID].pid;
+ if (pid->ns == &init_pid_ns)
+  if (move_pid_to_ns(pid, ns))
+   BUG();
+ set_task_vsession(tsk, pid->vnr);
+
+ pid = tsk->pids[PIDTYPE_PGID].pid;
+ if (pid->ns == &init_pid_ns)
+  if (move_pid_to_ns(pid, ns))
+   BUG();
+ set_task_vpgrp(tsk, pid->vnr);
+
+ ns->child_reaper = tsk;
+}
+#else
+struct pid_namespace *copy_pid_ns(int flags, struct pid_namespace *old_ns)
+{
+ BUG_ON(flags & CLONE_NEWPIDS);
+ return old_ns;
+}
+
+void move_init_to_ns(struct task_struct *tsk)
+{
+ BUG();
+}
+#endif
+
 /*
  * The pid hash table is scaled according to the amount of memory in the
  * machine.  From a minimum of 16 slots up to 4096 slots at one gigabyte or
```

Subject: [PATCH 9/13] Make proc be able to have multiple super blocks
Posted by Pavel Emelianov on Thu, 24 May 2007 13:01:24 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 bust hold the namespace it draws
the pids from.

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

```
diff --git a/include/linux/proc_fs.h b/include/linux/proc_fs.h
index 38f7082..d107a33 100644
--- a/include/linux/proc_fs.h
+++ b/include/linux/proc_fs.h
@@ -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 *);

 /*
diff --git a/fs/proc/inode.c b/fs/proc/inode.c
index 2ba47e4..5a7b5d5 100644
--- a/fs/proc/inode.c
+++ b/fs/proc/inode.c
@@ -15,6 +15,7 @@
 #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>
@@ -428,9 +429,17 @@ 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;
@@ -439,8 +448,8 @@ 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;
@@ -451,9 +460,10 @@ int proc_fill_super(struct super_block *
  return 0;
 out_no_root:
- printk("proc_read_super: get root inode failed\n");
  iput(root_inode);
- de_put(&proc_root);
+ de_put(root_dentry);
+out_no_de:
+ printk("proc_read_super: get root inode failed\n");
  return -ENOMEM;
 }
 MODULE_LICENSE("GPL");
diff --git a/fs/proc/internal.h b/fs/proc/internal.h
index 10f3601..b981956 100644
--- a/fs/proc/internal.h
+++ b/fs/proc/internal.h
@@ -71,3 +71,5 @@ static inline int proc_fd(struct inode *
 {
  return PROC_I(inode)->fd;
 }
+
+struct proc_dir_entry * create_proc_root(void);
diff --git a/fs/proc/root.c b/fs/proc/root.c
index 41f1703..e697f45 100644
--- a/fs/proc/root.c
+++ b/fs/proc/root.c
@@ -18,32 +18,76 @@
 #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)
+{
+ get_pid_ns((struct pid_namespace *)data);
+ 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 +197,24 @@ struct proc_dir_entry proc_root = {
  .parent  = &proc_root,
 };

+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);
```

Subject: [PATCH 10/13] Make proc draw pids from appropriate namespace
Posted by Pavel Emelianov on Thu, 24 May 2007 13:04:10 GMT

View Forum Message <> Reply to Message

The design is the following. The pids shown in proc tree are
the ones get from the namespace the superblock belongs to.

As seen from the previous patch, when proc is mounted the
current namespace is considered to be the owner of the superbloc.

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

```
diff --git a/fs/proc/base.c b/fs/proc/base.c
index 8b426e9..e9811ec 100644
--- a/fs/proc/base.c
+++ b/fs/proc/base.c
@@ -2105,6 +2106,7 @@ 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)
@@ -2114,8 +2116,9 @@ 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();
@@ -2132,7 +2135,8 @@ 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;
@@ -2140,9 +2144,9 @@ 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
@@ -2182,6 +2186,7 @@ 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;
@@ -2192,11 +2197,12 @@ 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);
@@ -2324,6 +2330,7 @@ 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;
@@ -2332,8 +2339,9 @@ 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();
@@ -2364,14 +2372,14 @@ 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;
  }
@@ -2440,6 +2448,7 @@ 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)
@@ -2473,12 +2482,13 @@ 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 */
```

Subject: [PATCH 11/13] Changes to show virtual ids to user
Posted by Pavel Emelianov on Thu, 24 May 2007 13:06:40 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.

An exception is copy_process - it was in one of the
previous patches - and the proc - this will come as a
separate patch.

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

```
diff --git a/arch/ia64/kernel/signal.c b/arch/ia64/kernel/signal.c
index aeec818..cdb64cc 100644
--- a/arch/ia64/kernel/signal.c
+++ b/arch/ia64/kernel/signal.c
@@ -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);
diff --git a/arch/parisc/kernel/signal.c b/arch/parisc/kernel/signal.c
index fb35ebc..2ce3806 100644
--- a/arch/parisc/kernel/signal.c
+++ b/arch/parisc/kernel/signal.c
@@ -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);
diff --git a/drivers/char/tty_io.c b/drivers/char/tty_io.c
index 260a1f3..9161107 100644
--- a/drivers/char/tty_io.c
+++ b/drivers/char/tty_io.c
@@ -103,6 +103,7 @@
 #include <linux/selection.h>

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

```
 #undef TTY_DEBUG_HANGUP

@@ -3065,7 +3066,7 @@ 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);
 }

 /**
@@ -3099,7 +3100,7 @@ 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;
@@ -3136,7 +3137,7 @@ 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);
 }

 /**
diff --git a/fs/binfmt_elf.c b/fs/binfmt_elf.c
index 7893feb..6f30d05 100644
--- a/fs/binfmt_elf.c
+++ b/fs/binfmt_elf.c
@@ -1325,10 +1325,10 @@ 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
@@ -1371,10 +1371,10 @@ 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;
diff --git a/fs/binfmt_elf_fdpic.c b/fs/binfmt_elf_fdpic.c
index 9bb9ff1..87d6eaf 100644
--- a/fs/binfmt_elf_fdpic.c
+++ b/fs/binfmt_elf_fdpic.c
@@ -1332,10 +1332,10 @@ 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
@@ -1381,10 +1381,10 @@ 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;
diff --git a/fs/exec.c b/fs/exec.c
index 0b68588..b8a3582 100644
--- a/fs/exec.c
+++ b/fs/exec.c
@@ -713,6 +713,9 @@ 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;
@@ -1301,7 +1304,7 @@ 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;
@@ -1373,7 +1376,7 @@ 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;
diff --git a/include/net/scm.h b/include/net/scm.h
index 5637d5e..43ed2f1 100644
--- a/include/net/scm.h
+++ b/include/net/scm.h
@@ -54,7 +54,7 @@ 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);
diff --git a/ipc/mqueue.c b/ipc/mqueue.c
```

```
index a242c83..9480a9e 100644
--- a/ipc/mqueue.c
+++ b/ipc/mqueue.c
@@ -513,7 +513,7 @@ 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,
diff --git a/ipc/msg.c b/ipc/msg.c
index 1cdb378..75fcbf6 100644
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -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);
diff --git a/ipc/sem.c b/ipc/sem.c
index 0f96683..5b2ef9a 100644
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -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();
diff --git a/ipc/shm.c b/ipc/shm.c
index bf28d5f..ae98d4f 100644
--- a/ipc/shm.c
+++ b/ipc/shm.c
@@ -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();
diff --git a/kernel/capability.c b/kernel/capability.c
index c8d3c77..a3d5395 100644
--- a/kernel/capability.c
+++ b/kernel/capability.c
@@ -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;
diff --git a/kernel/exit.c b/kernel/exit.c
index 43ce25b..942e01d 100644
--- a/kernel/exit.c
+++ b/kernel/exit.c
@@ -950,8 +950,8 @@ 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;
@@ -1047,13 +1047,13 @@ static int eligible_child(pid_t pid, int
  int err;

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

@@ -1126,7 +1126,7 @@ static int wait_task_zombie(struct task_
  int status;
  if (unlikely(noreap)) {
-  pid_t pid = p->pid;
+  pid_t pid = task_pid_nr_ns(p);
   uid_t uid = p->uid;
   int exit_code = p->exit_code;
   int why, status;
@@ -1244,7 +1244,7 @@ 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), &infop->si_pid);
  if (!retval && infop)
   retval = put_user(p->uid, &infop->si_uid);
  if (retval) {
@@ -1252,7 +1252,7 @@ static int wait_task_zombie(struct task_
   p->exit_state = EXIT_ZOMBIE;
   return retval;
  }
- retval = p->pid;
+ retval = task_pid_nr_ns(p);
  if (p->real_parent != p->parent) {
   write_lock_irq(&tasklist_lock);
   /* Double-check with lock held.  */
@@ -1312,7 +1312,7 @@ static int wait_task_stopped(struct task
  read_unlock(&tasklist_lock);

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

@@ -1383,11 +1383,11 @@ 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), &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);
  put_task_struct(p);

  BUG_ON(!retval);
```

```
@@ -1424,7 +1424,7 @@ static int wait_task_continued(struct ta
   p->signal->flags &= ~SIGNAL_STOP_CONTINUED;
  spin_unlock_irq(&p->sighand->siglock);

- pid = p->pid;
+ pid = task_pid_nr_ns(p);
  uid = p->uid;
  get_task_struct(p);
  read_unlock(&tasklist_lock);
@@ -1435,7 +1435,7 @@ 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);
  } else {
   retval = wait_noreap_copyout(p, pid, uid,
           CLD_CONTINUED, SIGCONT,
diff --git a/kernel/fork.c b/kernel/fork.c
index d7207a1..3ab517c 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -932,7 +932,7 @@ 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)
diff --git a/kernel/futex.c b/kernel/futex.c
index b7ce15c..4a2a46b 100644
--- a/kernel/futex.c
+++ b/kernel/futex.c
@@ -618,7 +618,7 @@ 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);

@@ -1334,7 +1334,7 @@ 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;
@@ -1719,7 +1719,7 @@ 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);
@@ -1729,7 +1729,7 @@ 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);
   /*
@@ -1759,7 +1759,7 @@ 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
@@ -1800,7 +1800,7 @@ 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();
@@ -1927,7 +1927,7 @@ 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:
@@ -1950,7 +1950,8 @@ retry_locked:
   */
  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();
  }

@@ -1960,7 +1961,7 @@ retry_locked:
   * 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;

  /*
@@ -2228,7 +2229,7 @@ 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
diff --git a/kernel/ptrace.c b/kernel/ptrace.c
index b1d11f1..11557c6 100644
--- a/kernel/ptrace.c
+++ b/kernel/ptrace.c
@@ -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);

diff --git a/kernel/sched.c b/kernel/sched.c
index 4784a8d..0c0a955 100644
--- a/kernel/sched.c
+++ b/kernel/sched.c
@@ -54,6 +54,7 @@
 #include <linux/kprobes.h>
 #include <linux/delayacct.h>
 #include <linux/reciprocal_div.h>
+#include <linux/pid_namespace.h>

 #include <asm/tlb.h>
 #include <asm/unistd.h>
@@ -1888,7 +1889,7 @@ 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);
 }

 /*
diff --git a/kernel/signal.c b/kernel/signal.c
index 75c5d77..0949043 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -663,7 +663,7 @@ 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:
@@ -1231,9 +1231,9 @@ static int kill_something_info(int sig,
   read_unlock(&tasklist_lock);
   ret = count ? retval : -ESRCH;
  } else if (pid < 0) {
-  ret = kill_pgrp_info(sig, info, find_pid(-pid));
+  ret = kill_pgrp_info(sig, info, find_vpid(-pid));
  } else {
-  ret = kill_pid_info(sig, info, find_pid(pid));
+  ret = kill_pid_info(sig, info, find_vpid(pid));
  }
  rcu_read_unlock();
  return ret;
@@ -1515,7 +1515,11 @@ 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. */
@@ -1580,7 +1584,11 @@ 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. */
@@ -1709,7 +1717,7 @@ 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.  */
@@ -1879,7 +1887,7 @@ 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;
   }

@@ -2268,7 +2276,7 @@ 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);
@@ -2284,12 +2292,12 @@ 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)) {
+ 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
diff --git a/kernel/sys.c b/kernel/sys.c
index 8aefd5e..890ff1c 100644
--- a/kernel/sys.c
+++ b/kernel/sys.c
@@ -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;
diff --git a/kernel/timer.c b/kernel/timer.c
index 00a38b8..a60eed1 100644
--- a/kernel/timer.c
+++ b/kernel/timer.c
@@ -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>
@@ -931,7 +932,7 @@ asmlinkage unsigned long sys_alarm(unsig
  */
 asmlinkage long sys_getpid(void)
 {
- return current->tgid;
+ return task_tgid_vnr(current);
 }

 /*
@@ -945,7 +946,7 @@ 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;
@@ -1077,7 +1078,7 @@ EXPORT_SYMBOL(schedule_timeout_uninterru
 /* Thread ID - the internal kernel "pid" */
 asmlinkage long sys_gettid(void)
 {
- return current->pid;
+ return task_pid_vnr(current);
 }

 /**
diff --git a/net/core/scm.c b/net/core/scm.c
index 292ad8d..63c6ea9 100644
--- a/net/core/scm.c
+++ b/net/core/scm.c
@@ -42,7 +42,7 @@

 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 ||
diff --git a/net/unix/af_unix.c b/net/unix/af_unix.c
index fc12ba5..261c344 100644
--- a/net/unix/af_unix.c
+++ b/net/unix/af_unix.c
@@ -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;
@@ -1069,7 +1069,7 @@ 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);
@@ -1133,7 +1133,7 @@ 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 12/13] Show appropriate pids in proc
Posted by Pavel Emelianov on Thu, 24 May 2007 13:08:13 GMT
View Forum Message <> Reply to Message

This is the proc-related part of the previous patch.
Since tasks are seen from two proc-s the appropriate
(virtual or global) pid must be shown.

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

```
diff --git a/fs/proc/array.c b/fs/proc/array.c
index aef7b7b..f65e4c9 100644
--- a/fs/proc/array.c
+++ b/fs/proc/array.c
@@ -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"
@@ -174,9 +177,12 @@ static inline char * task_state(struct t
   "Gid:\t%d\t%d\t%d\t%d\n",
   get_task_state(p),
   (p->sleep_avg/1024)*100/(1020000000/1024),
-        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);

@@ -349,6 +355,7 @@ 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);
@@ -381,9 +388,9 @@ static int do_task_stat(struct task_stru
    stime = cputime_add(stime, 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);
  }
@@ -414,7 +421,7 @@ 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),
   tcomm,
   state,
   ppid,
diff --git a/fs/proc/base.c b/fs/proc/base.c
index 8b426e9..e9811ec 100644
--- a/fs/proc/base.c
+++ b/fs/proc/base.c
@@ -73,6 +73,7 @@
 #include <linux/poll.h>
 #include <linux/nsproxy.h>
 #include <linux/oom.h>
+#include <linux/pid_namespace.h>
```

```
 #include "internal.h"

 /* NOTE:
@@ -1755,14 +1756,14 @@ 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));
 }
```

Subject: [PATCH 13/13] Make all proc entres accessible in a namespace
Posted by Pavel Emelianov on Thu, 24 May 2007 13:13:16 GMT

View Forum Message <> Reply to Message

This is a hack that must not go to any tree, but this
makes patch testing from shell possible.

When creating a new pid namespace one need to mount a new
proc instance in this space context. But the /proc will
contain pids and self symlink only and this breaks some
important tools like ps and sshd.

They expect /proc/stat, /proc/mounts and some other entries
to be present.

Without this patch the only possible way to test the
namespace is to make all the operations performed from
self-made programs. With it one may use general command
line tools.

This patch makes them visible, but this is ugly. Correct
fix must make some kind of proc_entries namespace but this
is another interesting task.

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

```
diff --git a/fs/proc/generic.c b/fs/proc/generic.c
index 9401a6f..eec1eb6 100644
--- a/fs/proc/generic.c
+++ b/fs/proc/generic.c
@@ -393,6 +393,10 @@ struct dentry *proc_lookup(struct inode
  spin_lock(&proc_subdir_lock);
  de = PDE(dir);
  if (de) {
+#ifdef CONFIG_PID_NS
+  if (de->parent == de)
+   de = &proc_root;
+#endif
   for (de = de->subdir; de ; de = de->next) {
    if (de->namelen != dentry->d_name.len)
     continue;
@@ -446,6 +450,10 @@ int proc_readdir(struct file * filp,
   ret = -EINVAL;
   goto out;
  }
+#ifdef CONFIG_PID_NS
+ if (de->parent == de)
+  de = &proc_root;
+#endif
  i = filp->f_pos;
  switch (i) {
   case 0:
```

Subject: Instructions of how to make testing easy
Posted by Pavel Emelianov on Thu, 24 May 2007 13:19:14 GMT

View Forum Message <> Reply to Message

That's the program I used for testing. It creates a new
session, chroots to new root, clones the namespace, mounts
proc and launches the sshd to keep track of the terminals.

The new root I prepared was bind-mounted /lib, /bin, /usr
etc directories, copied /dev devices with devpts mounted
inside and empty /var (for sshd) and /proc (for new mount).

After these preparations I launched this enter program and
then used ssh to get into the namespace.

Hope this will help.

The patches introduced was then tested with some mportaint
tests from ltp testsuite in 4 ways:

1 kernel w/o the patch
2 kernel with CONFIG_PID_NS=n
3 kernel with namespaces in init namespace
4 kernel with namespaces in subnamespace

The results coincided.

```
#include <unistd.h>
#include <sys/wait.h>
#include <errno.h>
#include <linux/unistd.h>

#ifndef __NR_unshare
#define __NR_unshare 310
#endif

_syscall1(int, unshare, int, flags)

#define CLONE_NEWPIDS 0x10000000
#define ROOT_DIR "./new_root"

int main(void)
{
 int pid;
 int status;

 pid = fork();
 if (pid < 0) {
  perror("Can't fork\n");
  return 1;
 }

 if (pid > 0) {
  if (waitpid(pid, &status, 0) < 0) {
   perror("Can't wait kid\n");
   return 2;
  }

  if (WIFEXITED(status))
   printf("%d exited with %d/%d\n", pid,
     WEXITSTATUS(status) & 0xf,
     WEXITSTATUS(status) >> 3);
  else if (WIFSIGNALED(status))
   printf("%d signalled with %d\n", pid, WTERMSIG(status));
  else
   printf("Some shit happened with %d\n", pid);
  return 0;
 }

 printf("Set new sid\n");
 if (setsid() < 0)
  return (errno << 4) + 0;

 printf("Unshare\n");
 if (unshare(CLONE_NEWPIDS) < 0)
  return (errno << 4) + 2;

 printf("Chroot\n");
 if (chroot(ROOT_DIR) < 0)
  return (errno << 4) + 1;

 printf("Mount proc\n");
 if (mount("none", "/proc", "proc", 0, NULL) < 0)
  return (errno << 4) + 3;

 printf("Launching sshd\n");
 if (fork() == 0) {
  execl("/usr/sbin/sshd", "/usr/sbin/sshd", "-p", "2202", NULL);
  return (errno << 4) + 3;
 }

 /* Never exit... Bad init */
 while (1) {
  if (wait(NULL) < 0)
   sleep(1);
 }
 return 0;
}
```

Subject: Re: Instructions of how to make testing easy
Posted by Cedric Le Goater on Thu, 24 May 2007 14:55:43 GMT
View Forum Message <> Reply to Message

Hello Pavel !

I'm giving it a try.

For those using qemu, you'll need this patch:

 http://lkml.org/lkml/2007/5/16/360

thanks for the patchset pavel.

C.

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by serue on Thu, 24 May 2007 15:09:09 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelianov (xemul@openvz.org):
> That's how OpenVZ sees the pid namespaces.
>
> The main idea is that kernel keeps operating with tasks pid
> as it did before, but each task obtains one more pid for each
> pid type - the virtual pid. When putting the pid to user or
> getting the pid from it kernel operates with the virtual ones.
>
> E.g. virtual pid is returned from getpid(), virtual pgid -
> from getpgid() and so on. Getting virtual pid from user is
> performed in setpgid(), setsid() and kill() mainly and in some
> other places.
>
> As far as the namespace are concerned I propose the following
> scheme. The namespace can be created from unshare syscall only.
> This makes fork() code look easier. Of course task must be

So is your main reason for posting this as a counter to Suka's patchset
the concern of overhead at clone?

thanks,
-serge

> prepared to have its pids changed. When task creates a new
> namespace it becomes its init and sees the tasks from it only.
> Tasks from init namespace see all the tasks.
>
> One relevant thing left behind is shrinking both proc's entries
> on task death. The reason I didn't do that is the following: this
> does not guarantee that the pid will be put (and thus still may
> hold the namespace), but makes the patch more complicated. So if
> this set will turns out to be interesting I will implement this
> thing as well.
>
> The patches are for 2.6.22-rc1-mm1 tree.
>
> Thanks,
> Pavel

Subject: Re: [PATCH 3/13] Introduciton of config option and clone flag
Posted by ebiederm on Thu, 24 May 2007 15:39:41 GMT
View Forum Message <> Reply to Message

Pavel Emelianov <xemul@openvz.org> writes:

```
>
> diff --git a/init/Kconfig b/init/Kconfig
> index 2a46e35..59e4625 100644
> --- a/init/Kconfig
> +++ b/init/Kconfig
> @@ -127,6 +127,16 @@ config SWAP_PREFETCH
>     Workstations and multiuser workstation servers will most likely want
>     to say Y.
>
> +config PID_NS
> + bool "Pid namespaces"
> + 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.
> +
```

Until we kill daemonize and are certain we can't find any other places
in the kernel that need to be updated please make PID_NS depend on
CONFIG_EXPERIMENTAL.

Eric

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by ebiederm on Thu, 24 May 2007 16:00:13 GMT
View Forum Message <> Reply to Message

Pavel Emelianov <xemul@openvz.org> writes:

> That's how OpenVZ sees the pid namespaces.
>
> The main idea is that kernel keeps operating with tasks pid
> as it did before, but each task obtains one more pid for each
> pid type - the virtual pid. When putting the pid to user or
> getting the pid from it kernel operates with the virtual ones.

Just a quick reaction.

- I would very much like to see a minimum of 3 levels of pids,
  being supported. Otherwise it is easy to overlook some of the
  cases that are required to properly support nesting, which long
  terms seems important.

- Semantically fork is easier then unshare. Unshare can mean
  a lot of things, and it is easy to pick a meaning that has weird
  side effects. Your implementation has a serious problem in that you
  change the value of getpid() at runtime. Glibc does not know how to
  cope with the value of getpid() changing.

Eric

Subject: Re: [PATCH 11/13] Changes to show virtual ids to user
Posted by ebiederm on Thu, 24 May 2007 16:05:59 GMT
View Forum Message <> Reply to Message

Pavel Emelianov <xemul@openvz.org> writes:

> 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.
>
> An exception is copy_process - it was in one of the
> previous patches - and the proc - this will come as a
> separate patch.

This is progress. However you don't currently handle the
case of sending a signal from one namespace to another or
passing unix credentials from one namespace to another.

In particular we need to know the pid of the source task
in the destination namespace.

Eric

Subject: Re: [PATCH 2/13] Small preparations for namespaces
Posted by serue on Thu, 24 May 2007 16:08:40 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelianov (xemul@openvz.org):
> This includes #ifdefs in get/put_pid_ns and rewriting
> the child_reaper() function to the more logical view.
>
> This doesn't fit logically into any other patch so
> I decided to make it separate.
>
> Signed-off-by: Pavel Emelianov <xemul@openvz.org>

```
>
> ---
>
> diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
> index 169c6c2..7af7191 100644
> --- a/include/linux/pid_namespace.h
> +++ b/include/linux/pid_namespace.h
> @@ -26,7 +26,9 @@ 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);
> @@ -34,12 +36,15 @@ 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 */
```

This can't be bisect-safe, right? You can't just use
tsk->nsproxy->pid_ns, as you've pointed out yourself.

-serge

Subject: Re: [PATCH 1/13] Round up the API
Posted by serue on Thu, 24 May 2007 16:09:46 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelianov (xemul@openvz.org):
> 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.
>
> I've sent this before, but Andrew didn't include it, so I resend it
> as the part of this set.
>
> Signed-off-by: Pavel Emelianov <xemul@openvz.org>
> Acked-by: Serge E. Hallyn <serue@us.ibm.com>
```

Yup, I still like this patch.

thanks,
-serge

```
>
> ---
>
> diff --git a/arch/mips/kernel/irixelf.c b/arch/mips/kernel/irixelf.c
> index 403d96f..10ba0a5 100644
> --- a/arch/mips/kernel/irixelf.c
> +++ b/arch/mips/kernel/irixelf.c
> @@ -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
> diff --git a/arch/mips/kernel/irixsig.c b/arch/mips/kernel/irixsig.c
> index 6980deb..210503e 100644
> --- a/arch/mips/kernel/irixsig.c
> +++ b/arch/mips/kernel/irixsig.c
> @@ -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;
> diff --git a/arch/mips/kernel/sysirix.c b/arch/mips/kernel/sysirix.c
> index 93a1484..23c3e82 100644
> --- a/arch/mips/kernel/sysirix.c
> +++ b/arch/mips/kernel/sysirix.c
> @@ -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;
> diff --git a/arch/sparc64/solaris/misc.c b/arch/sparc64/solaris/misc.c
> index 3b67de7..c86cb30 100644
> --- a/arch/sparc64/solaris/misc.c
> +++ b/arch/sparc64/solaris/misc.c
> @@ -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 */
>    {
> diff --git a/drivers/char/tty_io.c b/drivers/char/tty_io.c
> index 4251904..260a1f3 100644
> --- a/drivers/char/tty_io.c
> +++ b/drivers/char/tty_io.c
> @@ -3486,7 +3486,7 @@ 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);
> @@ -3496,7 +3496,7 @@ 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;
> diff --git a/fs/autofs/inode.c b/fs/autofs/inode.c
> index e7204d7..45f5992 100644
> --- a/fs/autofs/inode.c
> +++ b/fs/autofs/inode.c
> @@ -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;
>
> diff --git a/fs/autofs/root.c b/fs/autofs/root.c
> index c148953..592f640 100644
> --- a/fs/autofs/root.c
> +++ b/fs/autofs/root.c
> @@ -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)
> diff --git a/fs/autofs4/autofs_i.h b/fs/autofs4/autofs_i.h
> index d85f42f..2d4ae40 100644
> --- a/fs/autofs4/autofs_i.h
> +++ b/fs/autofs4/autofs_i.h
> @@ -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? */
> diff --git a/fs/autofs4/inode.c b/fs/autofs4/inode.c
> index 692364e..32a39b0 100644
> --- a/fs/autofs4/inode.c
> +++ b/fs/autofs4/inode.c
> @@ -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;
> diff --git a/fs/autofs4/root.c b/fs/autofs4/root.c
> index 2d4c8a3..c766ff8 100644
> --- a/fs/autofs4/root.c
> +++ b/fs/autofs4/root.c
> @@ -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)
> diff --git a/fs/binfmt_elf.c b/fs/binfmt_elf.c
> index fa8ea33..7893feb 100644
> --- a/fs/binfmt_elf.c
> +++ b/fs/binfmt_elf.c
> @@ -1327,8 +1327,8 @@ 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
> @@ -1373,8 +1373,8 @@ 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;
> diff --git a/fs/binfmt_elf_fdpic.c b/fs/binfmt_elf_fdpic.c
> index 9d62fba..9bb9ff1 100644
> --- a/fs/binfmt_elf_fdpic.c
> +++ b/fs/binfmt_elf_fdpic.c
> @@ -1334,8 +1334,8 @@ 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
> @@ -1383,8 +1383,8 @@ 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;
> diff --git a/fs/coda/upcall.c b/fs/coda/upcall.c
> index a5b5e63..3c35721 100644
> --- a/fs/coda/upcall.c
> +++ b/fs/coda/upcall.c
> @@ -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;
> diff --git a/fs/proc/array.c b/fs/proc/array.c
> index e798e11..aef7b7b 100644
> --- a/fs/proc/array.c
> +++ b/fs/proc/array.c
> @@ -381,8 +381,8 @@ static int do_task_stat(struct task_stru
>     stime = cputime_add(stime, 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);
> diff --git a/include/linux/sched.h b/include/linux/sched.h
> index 335dfc5..d4de6d8 100644
> --- a/include/linux/sched.h
> +++ b/include/linux/sched.h
> @@ -1092,24 +1092,19 @@ 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)
> +static inline pid_t task_session_nr(struct task_struct *tsk)
>  {
> - return sig->__session;
> + return tsk->signal->__session;
>  }
>
> -static inline pid_t process_session(struct task_struct *tsk)
> +static inline void set_task_session(struct task_struct *tsk, pid_t session)
>  {
> - return signal_session(tsk->signal);
> -}
> -
> -static inline void set_signal_session(struct signal_struct *sig, pid_t session)
> -{
> - sig->__session = session;
> + tsk->signal->__session = session;
>  }
>
>  static inline struct pid *task_pid(struct task_struct *task)
> diff --git a/kernel/exit.c b/kernel/exit.c
> index c6d14b8..43ce25b 100644
> --- a/kernel/exit.c
> +++ b/kernel/exit.c
> @@ -308,12 +308,12 @@ 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));
> @@ -1050,10 +1050,10 @@ 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;
>   }
>
> diff --git a/kernel/fork.c b/kernel/fork.c
> index 48928b1..d7207a1 100644
> --- a/kernel/fork.c
> +++ b/kernel/fork.c
> @@ -1249,8 +1249,8 @@ 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));
>
> diff --git a/kernel/signal.c b/kernel/signal.c
> index 3c09ee4..75c5d77 100644
> --- a/kernel/signal.c
> +++ b/kernel/signal.c
> @@ -506,7 +506,7 @@ 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))
> diff --git a/kernel/sys.c b/kernel/sys.c
> index e0e2da9..8aefd5e 100644
> --- a/kernel/sys.c
> +++ b/kernel/sys.c
> @@ -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;
```

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by xemul on Thu, 24 May 2007 16:11:30 GMT

View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelianov (xemul@openvz.org):
>> That's how OpenVZ sees the pid namespaces.
>>
>> The main idea is that kernel keeps operating with tasks pid
>> as it did before, but each task obtains one more pid for each
>> pid type - the virtual pid. When putting the pid to user or
>> getting the pid from it kernel operates with the virtual ones.
>>
>> E.g. virtual pid is returned from getpid(), virtual pgid -
>> from getpgid() and so on. Getting virtual pid from user is
>> performed in setpgid(), setsid() and kill() mainly and in some
>> other places.
>>
>> As far as the namespace are concerned I propose the following
>> scheme. The namespace can be created from unshare syscall only.
>> This makes fork() code look easier. Of course task must be
>
> So is your main reason for posting this as a counter to Suka's patchset
> the concern of overhead at clone?

No, that's just a coincidence that I worked on the same problem.

What I propose is another way to make pid namespaces. It has its
advantages over Suka's approach. Main are:

1. Lighter exporting of pid to userspace and performance issues
   on the whole - as you have noticed at least fork() is
   lighter and many syscalls that return task pids are;
2. Kernel logic of tracking pids is kept - virtual pids are
   used on kernel-user boundary only;
3. Cleaner logic for namespace migration: with this approach
   one need to save the virtual pid and let global one change;
   with Suka's logic this is not clear how to migrate the level
   2 namespace (concerning init to be level 0).

> thanks,
> -serge

Subject: Re: [PATCH 11/13] Changes to show virtual ids to user
Posted by xemul on Thu, 24 May 2007 16:15:40 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Pavel Emelianov <xemul@openvz.org> writes:
>
>> 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.
>>
>> An exception is copy_process - it was in one of the
>> previous patches - and the proc - this will come as a
>> separate patch.
>
>
> This is progress. However you don't currently handle the
> case of sending a signal from one namespace to another or
> passing unix credentials from one namespace to another.

That's true. Sending of signal from parent ns to children
is tricky question. It has many solutions, I wanted to
discuss which one is better:
1. Make an "unused" pid in each namespace and use it when signal
   comes from outside. This resembles the way it is done in OpenVZ.
2. Send the signal like it came from the kernel.

> In particular we need to know the pid of the source task
> in the destination namespace.

But the source task is not always visible in dst. In this case

we may use pid, that never exists in the destination, just like
it was kill run from bash by user.

> Eric
>

Subject: Re: Instructions of how to make testing easy
Posted by Pavel Emelianov on Thu, 24 May 2007 16:16:44 GMT
View Forum Message <> Reply to Message

Cedric Le Goater wrote:
> Hello Pavel !
>
> I'm giving it a try.

If you want, I can give you complete instructions
of how to do it. When I started to test it I faced
many tricky places and just want to make sure you
won't spent time solving the problems I solved.

> For those using qemu, you'll need this patch:
>
> http://lkml.org/lkml/2007/5/16/360
>
> thanks for the patchset pavel.
>
> C.
>
>

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by serue on Thu, 24 May 2007 16:20:06 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> Pavel Emelianov <xemul@openvz.org> writes:
>
> > That's how OpenVZ sees the pid namespaces.
> >
> > The main idea is that kernel keeps operating with tasks pid
> > as it did before, but each task obtains one more pid for each
> > pid type - the virtual pid. When putting the pid to user or
> > getting the pid from it kernel operates with the virtual ones.
>
> Just a quick reaction.

>
> - I would very much like to see a minimum of 3 levels of pids,
>   being supported. Otherwise it is easy to overlook some of the
>   cases that are required to properly support nesting, which long
>   terms seems important.

Pavel,

If I wanted to start a virtual server and in there start some checkpoint
restart jobs, so I start a new pid namespace inside the c/r job, what
will happen?

 a. second pidns unshare is refused
 b. second pidns unshare is allowed, but c/r job is not visible
 from the virtual server (but is from the global pidns)
 c. second pidns unshare is allowed, and somehow the c/r job
 is visible from the virtual server

If (a), is this a short-term shortcoming for simplicity of prototype and
code review, or do you think it's actually the right thing t do long
term?

thanks,
-serge

> - Semantically fork is easier then unshare. Unshare can mean
>   a lot of things, and it is easy to pick a meaning that has weird
>   side effects. Your implementation has a serious problem in that you
>   change the value of getpid() at runtime. Glibc does not know how to
>   cope with the value of getpid() changing.
>
> Eric

Subject: Re: [PATCH 1/13] Round up the API
Posted by ebiederm on Thu, 24 May 2007 16:22:45 GMT

View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Quoting Pavel Emelianov (xemul@openvz.org):
>> 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.
>>
>> I've sent this before, but Andrew didn't include it, so I resend it
>> as the part of this set.
>>
>> Signed-off-by: Pavel Emelianov <xemul@openvz.org>
>> Acked-by: Serge E. Hallyn <serue@us.ibm.com>
>
> Yup, I still like this patch.

I'm borderline. Less error prone interfaces sound good, less
duplication of information sounds good. Changing the names of
historical function may be change for the sake of change and
thus noise.

However if we are going to go this far I think we need to remove
the numeric pid cache from the task_struct.

Eric

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by Pavel Emelianov on Thu, 24 May 2007 16:26:10 GMT

View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Pavel Emelianov <xemul@openvz.org> writes:
>
>> That's how OpenVZ sees the pid namespaces.
>>
>> The main idea is that kernel keeps operating with tasks pid
>> as it did before, but each task obtains one more pid for each
>> pid type - the virtual pid. When putting the pid to user or
>> getting the pid from it kernel operates with the virtual ones.
>
> Just a quick reaction.
>
> - I would very much like to see a minimum of 3 levels of pids,

Why not 4? From my part, I would like to know, why such nesting
is important. We have plain IPC namespaces and nobody cares.
We will have isolated network namespaces, why pids are exception?

>   being supported. Otherwise it is easy to overlook some of the
>   cases that are required to properly support nesting, which long
>   terms seems important.
>
> - Semantically fork is easier then unshare. Unshare can mean

This is not. When you fork, the kid shares the session and the
group with its parent, but moving this pids to new ns is bad - the
parent will happen to be half-moved. Thus you need to break the
session and the group in fork(), but this is extra complexity.

>   a lot of things, and it is easy to pick a meaning that has weird
>   side effects. Your implementation has a serious problem in that you
>   change the value of getpid() at runtime. Glibc does not know how to
>   cope with the value of getpid() changing.

This pid changing happens only once per task lifetime. Though I haven't
seen any problems with glibc for many years running OpenVZ and I think,
that if glibc will want to cache this getpid() value we can teach it to
uncache this value in case someone called unshare() with CLONE_NEWPIDS.

> Eric
>

Thanks,
Pavel.

Subject: Re: [PATCH 1/13] Round up the API
Posted by Pavel Emelianov on Thu, 24 May 2007 16:31:41 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
>> Quoting Pavel Emelianov (xemul@openvz.org):
>>> 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.
>>>
>>> I've sent this before, but Andrew didn't include it, so I resend it
>>> as the part of this set.
>>>
>>> Signed-off-by: Pavel Emelianov <xemul@openvz.org>
>>> Acked-by: Serge E. Hallyn <serue@us.ibm.com>
>> Yup, I still like this patch.
>
> I'm borderline. Less error prone interfaces sound good, less
> duplication of information sounds good. Changing the names of
> historical function may be change for the sake of change and
> thus noise.

They are not historical. These calls appeared soon after new
struct pid subsystem.

>
> However if we are going to go this far I think we need to remove
> the numeric pid cache from the task_struct.

Object. Numerical pid and tgid on task makes it possible (and this
is done in ia64) to export this to user faster.

Moreover there can be places in kernel when we still hold the tasks
and want to know its pid, but the task is dead already and is going
to be delayed_put_task()-ed without pids aboard. I know this can
be properly if()-ed but what for?

> Eric
>

Subject: Re: [PATCH 11/13] Changes to show virtual ids to user
Posted by ebiederm on Thu, 24 May 2007 16:33:21 GMT
View Forum Message <> Reply to Message

Pavel Emelianov <xemul@sw.ru> writes:

> That's true. Sending of signal from parent ns to children
> is tricky question. It has many solutions, I wanted to
> discuss which one is better:

With unix domain sockets and the like it is conceivable we get
a pid transfer from one namespace to another and both namespaces
are leaf namespaces. I don't remember we can get a leaf to leaf
transfer when sending signals.

> 1. Make an "unused" pid in each namespace and use it when signal
>   comes from outside. This resembles the way it is done in OpenVZ.
> 2. Send the signal like it came from the kernel.
>
>> In particular we need to know the pid of the source task
>> in the destination namespace.
>
> But the source task is not always visible in dst. In this case
> we may use pid, that never exists in the destination, just like
> it was kill run from bash by user.

Quite true. So we have the question how do we name a the pid of
an unmapped task.

The two practical alternatives I see are:
- Map the struct pid into the namespace in question.
- Use pid == 0 (as if the kernel had generated the signal).
- Use pid == -1 (to signal an unknown user space task?)

My gut fee is that using pid == 0 is the simplest and most robust
way to handle it. That way we don't have information about things
outside the pid namespace leaking in. Of course I don't there may
be trust issues with reporting a user space process as pid == 0.

The worst case I can see with pid == 0. Is that it would be a bug
that we can fix later. For other cases it would seem to be a user
space API thing that we get stuck with for all time.

Eric

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by Daniel Lezcano on Thu, 24 May 2007 16:41:35 GMT

View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Eric W. Biederman wrote:
>
>> Pavel Emelianov <xemul@openvz.org> writes:
>>
>>
>>> That's how OpenVZ sees the pid namespaces.
>>>
>>> The main idea is that kernel keeps operating with tasks pid
>>> as it did before, but each task obtains one more pid for each
>>> pid type - the virtual pid. When putting the pid to user or
>>> getting the pid from it kernel operates with the virtual ones.
>>>

>> Just a quick reaction.
>>
>> - I would very much like to see a minimum of 3 levels of pids,
>>
>
> Why not 4? From my part, I would like to know, why such nesting
> is important. We have plain IPC namespaces and nobody cares.
> We will have isolated network namespaces, why pids are exception?
>
Pavel,

I am taking advantage to the opportunity to ask you if you plan to send
a new network namespace patchset ?

  -- Daniel

Subject: Re: [PATCH 1/13] Round up the API
Posted by serue on Thu, 24 May 2007 16:48:42 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> > Quoting Pavel Emelianov (xemul@openvz.org):
> >> 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.
> >>
> >> I've sent this before, but Andrew didn't include it, so I resend it
> >> as the part of this set.
> >>
> >> Signed-off-by: Pavel Emelianov <xemul@openvz.org>
> >> Acked-by: Serge E. Hallyn <serue@us.ibm.com>

> >
> > Yup, I still like this patch.
>
> I'm borderline. Less error prone interfaces sound good, less
> duplication of information sounds good. Changing the names of
> historical function may be change for the sake of change and
> thus noise.
>
> However if we are going to go this far I think we need to remove
> the numeric pid cache from the task_struct.

You mean tsk->pid?

I agree, especially in Suka's version. Not sure it applies to Pavel's
version, though since the "real"/global pid is still stored only in
tsk->pid, right?

-serge

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by serue on Thu, 24 May 2007 16:59:31 GMT

View Forum Message <> Reply to Message

Quoting Pavel Emelianov (xemul@sw.ru):
> Serge E. Hallyn wrote:
> > Quoting Pavel Emelianov (xemul@openvz.org):
> >> That's how OpenVZ sees the pid namespaces.
> >>
> >> The main idea is that kernel keeps operating with tasks pid
> >> as it did before, but each task obtains one more pid for each
> >> pid type - the virtual pid. When putting the pid to user or
> >> getting the pid from it kernel operates with the virtual ones.
> >>
> >> E.g. virtual pid is returned from getpid(), virtual pgid -
> >> from getpgid() and so on. Getting virtual pid from user is
> >> performed in setpgid(), setsid() and kill() mainly and in some
> >> other places.
> >>
> >> As far as the namespace are concerned I propose the following
> >> scheme. The namespace can be created from unshare syscall only.
> >> This makes fork() code look easier. Of course task must be
> >
> > So is your main reason for posting this as a counter to Suka's patchset
> > the concern of overhead at clone?
>
> No, that's just a coincidence that I worked on the same problem.
> What I propose is another way to make pid namespaces. It has its

> advantages over Suka's approach. Main are:
>
> 1. Lighter exporting of pid to userspace and performance issues
>   on the whole - as you have noticed at least fork() is
>   lighter and many syscalls that return task pids are;
> 2. Kernel logic of tracking pids is kept - virtual pids are
>   used on kernel-user boundary only;

On the other hand I've really learned to like the consistency of "there
is always a single active pid ns for the task from which it sees all
other tasks; it is seen in every pid ns for which it has a struct
upid."

> 3. Cleaner logic for namespace migration: with this approach
>   one need to save the virtual pid and let global one change;
>   with Suka's logic this is not clear how to migrate the level
>   2 namespace (concerning init to be level 0).

This is a very good point.

How *would* we migrate the pids at the second level?

-serge

Subject: Re: [PATCH 5/13] Expand the pid/task seeking functions set
Posted by Dave Hansen on Thu, 24 May 2007 17:11:04 GMT

View Forum Message <> Reply to Message

On Thu, 2007-05-24 at 16:50 +0400, Pavel Emelianov wrote:

```
>
> +struct pid * fastcall __find_vpid(int nr, struct pid_namespace *ns)
> +{
> +#ifdef CONFIG_PID_NS
> +	struct hlist_node *elem;
> +	struct pid *pid;
> +#endif
> +
> +	if (ns == &init_pid_ns)
> +		return find_pid(nr);
> +
> +#ifdef CONFIG_PID_NS
> +	hlist_for_each_entry_rcu(pid, elem,
> +			&vpid_hash[vpid_hashfn(nr, ns)], vpid_chain) {
> +		if (pid->vnr == nr && pid->ns == ns)
> +			return pid;
> +	}
> +#endif
```

> +	return NULL;
> +}

I am a bit worried that there are too many #ifdefs here. Your patch
series adds ~20 of them, and they look to me to be mostly in .c files.
Section 2 in SubmittingPatches somewhat discourages this.

Do you have any plans for cleaning these up?

-- Dave

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

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by ebiederm on Thu, 24 May 2007 18:56:53 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

>> 3. Cleaner logic for namespace migration: with this approach
>>   one need to save the virtual pid and let global one change;
>>   with Suka's logic this is not clear how to migrate the level
>>   2 namespace (concerning init to be level 0).
>
> This is a very good point.
>
> How *would* we migrate the pids at the second level?

As long as you don't try and restore pids into the initial pid namespace
it isn't a problem. You just record the pid hierarchy and the pid
for a task in that hierarchy. There really is nothing special going on
that should make migration hard.

Or did I miss something?

Eric

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by serue on Thu, 24 May 2007 19:10:57 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):

> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> >> 3. Cleaner logic for namespace migration: with this approach
> >>   one need to save the virtual pid and let global one change;
> >>   with Suka's logic this is not clear how to migrate the level
> >>   2 namespace (concerning init to be level 0).
> >
> > This is a very good point.
> >
> > How *would* we migrate the pids at the second level?
>
> As long as you don't try and restore pids into the initial pid namespace
> it isn't a problem. You just record the pid hierarchy and the pid
> for a task in that hierarchy. There really is nothing special going on
> that should make migration hard.
>
> Or did I miss something?

Hmm, no, i guess you are right. I was thinking that getting the pid for
a process woudl be done purely from userspace, but I guess along with a
kernel helper to *set* pids, we could also have a kernel helper to get
all pids for all pid namespaces "above" that of the process doing the
checkpoint.

Makes sense.

thanks,
-serge

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by ebiederm on Thu, 24 May 2007 19:18:48 GMT

View Forum Message <> Reply to Message

Pavel Emelianov <xemul@openvz.org> writes:

> Eric W. Biederman wrote:
>> Pavel Emelianov <xemul@openvz.org> writes:
>>
>>> That's how OpenVZ sees the pid namespaces.
>>>
>>> The main idea is that kernel keeps operating with tasks pid
>>> as it did before, but each task obtains one more pid for each
>>> pid type - the virtual pid. When putting the pid to user or
>>> getting the pid from it kernel operates with the virtual ones.
>>
>> Just a quick reaction.
>>

>> - I would very much like to see a minimum of 3 levels of pids,
>
> Why not 4? From my part, I would like to know, why such nesting
> is important. We have plain IPC namespaces and nobody cares.
> We will have isolated network namespaces, why pids are exception?

4+ is fine, and something we will probably care about someday.
3 seems to be the minimum necessary to get people thinking about
adding more so we don't have arbitrary special cases, especially
in the user interface. At 3 the things are simple enough we don't
have to allocate additional data structures etc.

If we don't need nesting we don't even need 2 levels, and we
can remove the global pid. But we have had that conversation
and especially for the current OpenVZ usage we need nesting.

Having more then two layers means we are prepared to use pid namespaces more
generally. It really isn't that much harder.

>>   being supported. Otherwise it is easy to overlook some of the
>>   cases that are required to properly support nesting, which long
>>   terms seems important.
>>
>> - Semantically fork is easier then unshare. Unshare can mean
>
> This is not. When you fork, the kid shares the session and the
> group with its parent, but moving this pids to new ns is bad - the
> parent will happen to be half-moved. Thus you need to break the
> session and the group in fork(), but this is extra complexity.

Nope. You will just need to have the child call setsid() if
you don't want to share the session and the group.

You can perfectly well share the sid and group with the parent,
because internal to the kernel pids aren't numeric, they are struct
pid pointers.

There is the question of do you use foreign pid handling to display
the session and the group, or do you allocate pids for the session
and the group in the new pid namespace. At this point foreign pid
handling looks sufficient.

>>   a lot of things, and it is easy to pick a meaning that has weird
>>   side effects. Your implementation has a serious problem in that you
>>   change the value of getpid() at runtime. Glibc does not know how to
>>   cope with the value of getpid() changing.
>
> This pid changing happens only once per task lifetime.

Unshare isn't once per task lifetime, unless you added some extra
constraints.

> Though I haven't
> seen any problems with glibc for many years running OpenVZ and I think,
> that if glibc will want to cache this getpid() value we can teach it to
> uncache this value in case someone called unshare() with CLONE_NEWPIDS.

glibc very much caches the results of getpid().

If you want to teach glibc not to cache getpid() fee free. The only
way I know to get glibc to invalidates it's pid cache is to call fork.

Eric

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by Pavel Emelianov on Fri, 25 May 2007 06:29:42 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Eric W. Biederman (ebiederm@xmission.com):
>> Pavel Emelianov <xemul@openvz.org> writes:
>>
>>> That's how OpenVZ sees the pid namespaces.
>>>
>>> The main idea is that kernel keeps operating with tasks pid
>>> as it did before, but each task obtains one more pid for each
>>> pid type - the virtual pid. When putting the pid to user or
>>> getting the pid from it kernel operates with the virtual ones.
>> Just a quick reaction.
>>
>> - I would very much like to see a minimum of 3 levels of pids,
>> being supported. Otherwise it is easy to overlook some of the
>> cases that are required to properly support nesting, which long
>> terms seems important.
>
> Pavel,
>
> If I wanted to start a virtual server and in there start some checkpoint
> restart jobs, so I start a new pid namespace inside the c/r job, what
> will happen?

What will happen with this namespace on restore? What pids will
you assign to it in the parent (but not that init) namespace?

a. arbitrary: that means that you don't care that subgroup

   of tasks in the VS namespace. Thus why don't move them
   into separate namespace
b. try to hold them as they were: this way is likely to fail
   and can work w/o namespaces at all.

So what's your answer?

> a. second pidns unshare is refused
> b. second pidns unshare is allowed, but c/r job is not visible
> from the virtual server (but is from the global pidns)
> c. second pidns unshare is allowed, and somehow the c/r job
> is visible from the virtual server
>
> If (a), is this a short-term shortcoming for simplicity of prototype and
> code review, or do you think it's actually the right thing t do long
> term?
>
> thanks,
> -serge
>
>> - Semantically fork is easier then unshare. Unshare can mean
>>   a lot of things, and it is easy to pick a meaning that has weird
>>   side effects. Your implementation has a serious problem in that you
>>   change the value of getpid() at runtime. Glibc does not know how to
>>   cope with the value of getpid() changing.
>>
>> Eric
>

Subject: Re: [PATCH 11/13] Changes to show virtual ids to user
Posted by Pavel Emelianov on Fri, 25 May 2007 06:33:22 GMT

View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Pavel Emelianov <xemul@sw.ru> writes:
>
>> That's true. Sending of signal from parent ns to children
>> is tricky question. It has many solutions, I wanted to
>> discuss which one is better:
>
> With unix domain sockets and the like it is conceivable we get
> a pid transfer from one namespace to another and both namespaces
> are leaf namespaces. I don't remember we can get a leaf to leaf
> transfer when sending signals.

We should not allow any transfer from leaf NS to leaf NS.
Should I explain why?

>> 1. Make an "unused" pid in each namespace and use it when signal
>>   comes from outside. This resembles the way it is done in OpenVZ.
>> 2. Send the signal like it came from the kernel.
>>
>>> In particular we need to know the pid of the source task
>>> in the destination namespace.
>> But the source task is not always visible in dst. In this case
>> we may use pid, that never exists in the destination, just like
>> it was kill run from bash by user.
>
> Quite true. So we have the question how do we name a the pid of
> an unmapped task.
>
> The two practical alternatives I see are:
> - Map the struct pid into the namespace in question.

Bad solution. We will poison the dst namespace

> - Use pid == 0 (as if the kernel had generated the signal).

Not just pid 0, but SI_KERNEL in si_code.

> - Use pid == -1 (to signal an unknown user space task?)

Hm... Strange solution.

> My gut fee is that using pid == 0 is the simplest and most robust
> way to handle it. That way we don't have information about things
> outside the pid namespace leaking in. Of course I don't there may
> be trust issues with reporting a user space process as pid == 0.
>
> The worst case I can see with pid == 0. Is that it would be a bug
> that we can fix later. For other cases it would seem to be a user
> space API thing that we get stuck with for all time.

We cannot trust userspace application to expect some pid other than
positive. All that we can is either use some always-absent pid or
send the signal as SI_KERNEL.

Our experience show that making decisions like above causes random
applications failures that are hard (or even impossible) to debug.

> Eric
>

Subject: Re: [PATCH 2/13] Small preparations for namespaces
Posted by Pavel Emelianov on Fri, 25 May 2007 06:58:20 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
```
> Quoting Pavel Emelianov (xemul@openvz.org):
>> This includes #ifdefs in get/put_pid_ns and rewriting
>> the child_reaper() function to the more logical view.
>>
>> This doesn't fit logically into any other patch so
>> I decided to make it separate.
>>
>> Signed-off-by: Pavel Emelianov <xemul@openvz.org>
>>
>> ---
>>
>> diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
>> index 169c6c2..7af7191 100644
>> --- a/include/linux/pid_namespace.h
>> +++ b/include/linux/pid_namespace.h
>> @@ -26,7 +26,9 @@ 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);
>> @@ -34,12 +36,15 @@ 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 */
>
> This can't be bisect-safe, right?  You can't just use
```

> tsk->nsproxy->pid_ns, as you've pointed out yourself.

I can :) See - I have a proving BUG_ON() here.

>
> -serge
>

Subject: Re: [PATCH 1/13] Round up the API
Posted by Pavel Emelianov on Fri, 25 May 2007 07:00:21 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Eric W. Biederman (ebiederm@xmission.com):
>> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>>
>>> Quoting Pavel Emelianov (xemul@openvz.org):
>>>> 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.
>>>>
>>>> I've sent this before, but Andrew didn't include it, so I resend it
>>>> as the part of this set.
>>>>
>>>> Signed-off-by: Pavel Emelianov <xemul@openvz.org>
>>>> Acked-by: Serge E. Hallyn <serue@us.ibm.com>
>>> Yup, I still like this patch.
>> I'm borderline. Less error prone interfaces sound good, less
>> duplication of information sounds good. Changing the names of
>> historical function may be change for the sake of change and
>> thus noise.
>>
>> However if we are going to go this far I think we need to remove
>> the numeric pid cache from the task_struct.
>
> You mean tsk->pid?
>

> I agree, especially in Suka's version. Not sure it applies to Pavel's
> version, though since the "real"/global pid is still stored only in
> tsk->pid, right?

No. All objects that have pid (task_struct, signal_struct and pid (struct))
have two ids after this patch - virtual one and global one.

> -serge
>

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by Pavel Emelianov on Fri, 25 May 2007 07:06:49 GMT

View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelianov (xemul@sw.ru):
>> Serge E. Hallyn wrote:
>>> Quoting Pavel Emelianov (xemul@openvz.org):
>>>> That's how OpenVZ sees the pid namespaces.
>>>>
>>>> The main idea is that kernel keeps operating with tasks pid
>>>> as it did before, but each task obtains one more pid for each
>>>> pid type - the virtual pid. When putting the pid to user or
>>>> getting the pid from it kernel operates with the virtual ones.
>>>>
>>>> E.g. virtual pid is returned from getpid(), virtual pgid -
>>>> from getpgid() and so on. Getting virtual pid from user is
>>>> performed in setpgid(), setsid() and kill() mainly and in some
>>>> other places.
>>>>
>>>> As far as the namespace are concerned I propose the following
>>>> scheme. The namespace can be created from unshare syscall only.
>>>> This makes fork() code look easier. Of course task must be
>>> So is your main reason for posting this as a counter to Suka's patchset
>>> the concern of overhead at clone?
>> No, that's just a coincidence that I worked on the same problem.
>> What I propose is another way to make pid namespaces. It has its
>> advantages over Suka's approach. Main are:
>>
>> 1. Lighter exporting of pid to userspace and performance issues
>>   on the whole - as you have noticed at least fork() is
>>   lighter and many syscalls that return task pids are;
>> 2. Kernel logic of tracking pids is kept - virtual pids are
>>   used on kernel-user boundary only;
>
> On the other hand I've really learned to like the consistency of "there
> is always a single active pid ns for the task from which it sees all

> other tasks; it is seen in every pid ns for which it has a struct
> upid."
>
>> 3. Cleaner logic for namespace migration: with this approach
>>   one need to save the virtual pid and let global one change;
>>   with Suka's logic this is not clear how to migrate the level
>>   2 namespace (concerning init to be level 0).
>
> This is a very good point.
>
> How *would* we migrate the pids at the second level?

I would like to know how you migrate a *part* of a virtual
server? What happens with pids, IPC ids, network connections?

There are many entities in VS that are not bound to task, but to
VS and if you migrate only half of them you're risking in loosing
the integrity of the VS. If you don't care it - why do you need
namespaces at all?

> -serge
>

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by Pavel Emelianov on Fri, 25 May 2007 07:08:31 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Eric W. Biederman (ebiederm@xmission.com):
>> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>>
>>>> 3. Cleaner logic for namespace migration: with this approach
>>>>   one need to save the virtual pid and let global one change;
>>>>   with Suka's logic this is not clear how to migrate the level
>>>>   2 namespace (concerning init to be level 0).
>>> This is a very good point.
>>>
>>> How *would* we migrate the pids at the second level?
>> As long as you don't try and restore pids into the initial pid namespace
>> it isn't a problem. You just record the pid hierarchy and the pid
>> for a task in that hierarchy. There really is nothing special going on
>> that should make migration hard.
>>
>> Or did I miss something?
>
> Hmm, no, i guess you are right. I was thinking that getting the pid for
> a process woudl be done purely from userspace, but I guess along with a

> kernel helper to *set* pids, we could also have a kernel helper to get
> all pids for all pid namespaces "above" that of the process doing the
> checkpoint.

So do you agree that if we migrate a VS we need to migrate the whole VS?

> Makes sense.
>
> thanks,
> -serge
>

Subject: Re: [PATCH 5/13] Expand the pid/task seeking functions set
Posted by Pavel Emelianov on Fri, 25 May 2007 07:08:48 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Thu, 2007-05-24 at 16:50 +0400, Pavel Emelianov wrote:
```
>> +struct pid * fastcall __find_vpid(int nr, struct pid_namespace *ns)
>> +{
>> +#ifdef CONFIG_PID_NS
>> +	struct hlist_node *elem;
>> +	struct pid *pid;
>> +#endif
>> +
>> +	if (ns == &init_pid_ns)
>> +		return find_pid(nr);
>> +
>> +#ifdef CONFIG_PID_NS
>> +	hlist_for_each_entry_rcu(pid, elem,
>> +			&vpid_hash[vpid_hashfn(nr, ns)], vpid_chain) {
>> +		if (pid->vnr == nr && pid->ns == ns)
>> +			return pid;
>> +	}
>> +#endif
>> +	return NULL;
>> +}
```
>
> I am a bit worried that there are too many #ifdefs here. Your patch
> series adds ~20 of them, and they look to me to be mostly in .c files.
> Section 2 in SubmittingPatches somewhat discourages this.
>
> Do you have any plans for cleaning these up?

Sure I have. But this approach makes review simpler - everyone
explicitly see what exact actions are taken in each place. In
the second iteration this will be make in a more elegant way

like making static inline stubs etc.

This set is a kind of RFC and proof-of-concept. I didn't intent
this to be merged to any tree as is. That's why a attached the
lats patch with strut in proc to observe the whole tree.

BTW, question to Sukadev - how did you test your patches? I do
know that ps utility doesn't work without full /proc tree and
I don's see similar hacks in your patchset.

> -- Dave
>
>

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

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by Pavel Emelianov on Fri, 25 May 2007 07:15:51 GMT

View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Pavel Emelianov <xemul@openvz.org> writes:
>
>> Eric W. Biederman wrote:
>>> Pavel Emelianov <xemul@openvz.org> writes:
>>>
>>>> That's how OpenVZ sees the pid namespaces.
>>>>
>>>> The main idea is that kernel keeps operating with tasks pid
>>>> as it did before, but each task obtains one more pid for each
>>>> pid type - the virtual pid. When putting the pid to user or
>>>> getting the pid from it kernel operates with the virtual ones.
>>> Just a quick reaction.
>>>
>>> - I would very much like to see a minimum of 3 levels of pids,
>> Why not 4? From my part, I would like to know, why such nesting
>> is important. We have plain IPC namespaces and nobody cares.
>> We will have isolated network namespaces, why pids are exception?
>
> 4+ is fine, and something we will probably care about someday.
> 3 seems to be the minimum necessary to get people thinking about
> adding more so we don't have arbitrary special cases, especially
> in the user interface. At 3 the things are simple enough we don't
> have to allocate additional data structures etc.

>
> If we don't need nesting we don't even need 2 levels, and we
> can remove the global pid. But we have had that conversation
> and especially for the current OpenVZ usage we need nesting.

We need nesting but 2 levels is more than enough. Yet again -
we have 2 level IPC namespace, 2 level network namespace etc.

Generic structures are not always needed. Say, why don't we
have N-level page tables in kernel? Why not make them generic?
What if some ia128 architecture will require 7-level tables!?

> Having more then two layers means we are prepared to use pid namespaces more
> generally. It really isn't that much harder.

It is not, but do we need to spend so much time on solving
not relevant problems?

>>>   being supported. Otherwise it is easy to overlook some of the
>>>   cases that are required to properly support nesting, which long
>>>   terms seems important.
>>>
>>> - Semantically fork is easier then unshare. Unshare can mean
>> This is not. When you fork, the kid shares the session and the
>> group with its parent, but moving this pids to new ns is bad - the
>> parent will happen to be half-moved. Thus you need to break the
>> session and the group in fork(), but this is extra complexity.
>
> Nope. You will just need to have the child call setsid() if
> you don't want to share the session and the group.

Of course, but setsid() must be done *before* creating a new
namespace, Otherwise you will have a half-inserted into new
namespace task. This sounds awful.

> You can perfectly well share the sid and group with the parent,
> because internal to the kernel pids aren't numeric, they are struct
> pid pointers.
>
> There is the question of do you use foreign pid handling to display
> the session and the group, or do you allocate pids for the session
> and the group in the new pid namespace. At this point foreign pid
> handling looks sufficient.
>
>>>   a lot of things, and it is easy to pick a meaning that has weird
>>>   side effects. Your implementation has a serious problem in that you
>>>   change the value of getpid() at runtime. Glibc does not know how to
>>>   cope with the value of getpid() changing.

>> This pid changing happens only once per task lifetime.
>
> Unshare isn't once per task lifetime, unless you added some extra
> constraints.

It is once. You create a new namespace and that's all.

>> Though I haven't
>> seen any problems with glibc for many years running OpenVZ and I think,
>> that if glibc will want to cache this getpid() value we can teach it to
>> uncache this value in case someone called unshare() with CLONE_NEWPIDS.
>
> glibc very much caches the results of getpid().

Can you prove it? We have run OpenVZ for many years and with many
userspace configurations and we haven't seen the problems with
glibc ever.

> If you want to teach glibc not to cache getpid() fee free. The only
> way I know to get glibc to invalidates it's pid cache is to call fork.
>
> Eric
>

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by Pavel Emelianov on Fri, 25 May 2007 07:26:17 GMT

View Forum Message <> Reply to Message

Daniel Lezcano wrote:
> Pavel Emelianov wrote:
>> Eric W. Biederman wrote:
>>
>>> Pavel Emelianov <xemul@openvz.org> writes:
>>>
>>>
>>>> That's how OpenVZ sees the pid namespaces.
>>>>
>>>> The main idea is that kernel keeps operating with tasks pid
>>>> as it did before, but each task obtains one more pid for each
>>>> pid type - the virtual pid. When putting the pid to user or
>>>> getting the pid from it kernel operates with the virtual ones.
>>>>
>>> Just a quick reaction.
>>> - I would very much like to see a minimum of 3 levels of pids,
>>>
>>
>> Why not 4? From my part, I would like to know, why such nesting

>> is important. We have plain IPC namespaces and nobody cares.
>> We will have isolated network namespaces, why pids are exception?
>>
> Pavel,
>
> I am taking advantage to the opportunity to ask you if you plan to send
> a new network namespace patchset ?

Unfortunately no :(Right now we're focusing on pids and
resource management.

> -- Daniel
>

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by Daniel Lezcano on Fri, 25 May 2007 08:30:43 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Daniel Lezcano wrote:
>
>> Pavel Emelianov wrote:
>>
>>> Eric W. Biederman wrote:
>>>
>>>
>>>> Pavel Emelianov <xemul@openvz.org> writes:
>>>>
>>>>
>>>>
>>>>> That's how OpenVZ sees the pid namespaces.
>>>>>
>>>>> The main idea is that kernel keeps operating with tasks pid
>>>>> as it did before, but each task obtains one more pid for each
>>>>> pid type - the virtual pid. When putting the pid to user or
>>>>> getting the pid from it kernel operates with the virtual ones.
>>>>>
>>>>>
>>>> Just a quick reaction.
>>>> - I would very much like to see a minimum of 3 levels of pids,
>>>>
>>>>

>>> Why not 4? From my part, I would like to know, why such nesting
>>> is important. We have plain IPC namespaces and nobody cares.
>>> We will have isolated network namespaces, why pids are exception?
>>>
>>>
>> Pavel,
>>
>> I am taking advantage to the opportunity to ask you if you plan to send
>> a new network namespace patchset ?
>>
>
> Unfortunately no :(Right now we're focusing on pids and
> resource management.
>
Yep, a big part :)

Did you, OpenVZ guys, had time to look at Eric's patchset ?

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

Subject: Re: [PATCH 2/13] Small preparations for namespaces
Posted by serue on Fri, 25 May 2007 13:01:18 GMT

View Forum Message <> Reply to Message

Quoting Pavel Emelianov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > Quoting Pavel Emelianov (xemul@openvz.org):
> >> This includes #ifdefs in get/put_pid_ns and rewriting
> >> the child_reaper() function to the more logical view.
> >>
> >> This doesn't fit logically into any other patch so
> >> I decided to make it separate.
> >>
> >> Signed-off-by: Pavel Emelianov <xemul@openvz.org>
> >>
> >> ---
> >>
> >> diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
> >> index 169c6c2..7af7191 100644
> >> --- a/include/linux/pid_namespace.h
> >> +++ b/include/linux/pid_namespace.h
> >> @@ -26,7 +26,9 @@ 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);
> >> @@ -34,12 +36,15 @@ 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 */
> >
> > This can't be bisect-safe, right?  You can't just use
> > tsk->nsproxy->pid_ns, as you've pointed out yourself.
>
> I can :) See - I have a proving BUG_ON() here.
```

I didn't know BUG_ON()'s actually warded off bugs :)

You've tested this with the infamous NFS testcase?

I don't see *why* it would work for you, but if you claim it does, I
guess you'd know better than I :)

-serge

Subject: Re: [PATCH 1/13] Round up the API
Posted by serue on Fri, 25 May 2007 13:02:30 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelianov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > Quoting Eric W. Biederman (ebiederm@xmission.com):

> >> "Serge E. Hallyn" <serue@us.ibm.com> writes:
> >>
> >>> Quoting Pavel Emelianov (xemul@openvz.org):
> >>>> 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.
> >>>>
> >>>> I've sent this before, but Andrew didn't include it, so I resend it
> >>>> as the part of this set.
> >>>>
> >>>> Signed-off-by: Pavel Emelianov <xemul@openvz.org>
> >>>> Acked-by: Serge E. Hallyn <serue@us.ibm.com>
> >>> Yup, I still like this patch.
> >> I'm borderline. Less error prone interfaces sound good, less
> >> duplication of information sounds good. Changing the names of
> >> historical function may be change for the sake of change and
> >> thus noise.
> >>
> >> However if we are going to go this far I think we need to remove
> >> the numeric pid cache from the task_struct.
> >
> > You mean tsk->pid?
> >
> > I agree, especially in Suka's version. Not sure it applies to Pavel's
> > version, though since the "real"/global pid is still stored only in
> > tsk->pid, right?
>
> No. All objects that have pid (task_struct, signal_struct and pid (struct))
> have two ids after this patch - virtual one and global one.

(Yes, so wouldn't removing task->pid be pretty detrimental?)

Could you outline how you would extend this to 3 levels? Would you just
add a 'vpid2' etc to the struct pid?

thanks,
-serge

Subject: Re: [PATCH 2/13] Small preparations for namespaces
Posted by Pavel Emelianov on Fri, 25 May 2007 13:21:12 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelianov (xemul@openvz.org):
>> Serge E. Hallyn wrote:
>>> Quoting Pavel Emelianov (xemul@openvz.org):
>>>> This includes #ifdefs in get/put_pid_ns and rewriting
>>>> the child_reaper() function to the more logical view.
>>>>
>>>> This doesn't fit logically into any other patch so
>>>> I decided to make it separate.
>>>>
>>>> Signed-off-by: Pavel Emelianov <xemul@openvz.org>
>>>>

```
>>>> ---
>>>>
>>>> diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
>>>> index 169c6c2..7af7191 100644
>>>> --- a/include/linux/pid_namespace.h
>>>> +++ b/include/linux/pid_namespace.h
>>>> @@ -26,7 +26,9 @@ 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);
>>>> @@ -34,12 +36,15 @@ 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 */
```

>>> This can't be bisect-safe, right? You can't just use
>>> tsk->nsproxy->pid_ns, as you've pointed out yourself.
>> I can :) See - I have a proving BUG_ON() here.
>
> I didn't know BUG_ON()'s actually warded off bugs :)

It does not, but it says to code reader that this call
expects something special. In this case - tsk is expected
to be current always. And it is.

> You've tested this with the infamous NFS testcase?

What testcase do you mean?

> I don't see *why* it would work for you, but if you claim it does, I
> guess you'd know better than I :)

I don't get you here. I've checked that the task passed to
child_reaper is current always. This BUG_ON prevents later
code from passing arbitrary task to it.

> -serge
>

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by serue on Fri, 25 May 2007 13:25:18 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelianov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > Quoting Eric W. Biederman (ebiederm@xmission.com):
> >> "Serge E. Hallyn" <serue@us.ibm.com> writes:
> >>
> >>>> 3. Cleaner logic for namespace migration: with this approach
> >>>>   one need to save the virtual pid and let global one change;
> >>>>   with Suka's logic this is not clear how to migrate the level
> >>>>   2 namespace (concerning init to be level 0).
> >>> This is a very good point.
> >>>
> >>> How *would* we migrate the pids at the second level?
> >> As long as you don't try and restore pids into the initial pid namespace
> >> it isn't a problem. You just record the pid hierarchy and the pid
> >> for a task in that hierarchy. There really is nothing special going on
> >> that should make migration hard.
> >>
> >> Or did I miss something?
> >

> > Hmm, no, i guess you are right. I was thinking that getting the pid for
> > a process woudl be done purely from userspace, but I guess along with a
> > kernel helper to *set* pids, we could also have a kernel helper to get
> > all pids for all pid namespaces "above" that of the process doing the
> > checkpoint.
>
> So do you agree that if we migrate a VS we need to migrate the whole VS?

I started to respond, then realized you were probably asking something
different than I thought. My original response is below, but here is I
think the answer to your question, which is important because I think
your question might highlight a misunderstanding about the design of
Suka's code.

Let's say a vserver is started, and in there a pidns is started for a
checkpoint/restart job. So let's say we have PID 13 in the root
namespace starting PID 14 in a new namespace. So using (pid, pid_ns) as
the terminology, we havd (13,1) as the parent process, and (14,1)=(1,2)
as the init of the vserver. Let's ignore other tasks inthe vserver, and
just talk about (1402,2) as the init of the checkpoint restart job, so
it is (1402,2)=(1,3). And oh, yeah, (1402,2)=(1,3)=(2309,1).

Now when we want to migrate the vserver, a task in pid_ns 2 will look
for all tasks with pids in pidns 2. That will automatically include all
tasks in pid_ns 3. I think you thought I was asking how we would
include pid_ns 3, and are asking whether it woudl be ok to not migrate
pid_ns 3? (answer: it's irrelevant, all tasks in pid_ns 3 are also in
pid_ns 2 - and in pid_ns 1).

What I was actually asking was, in the same situation, how would the
task in pid_ns 2 doing the checkpoint get the pids in pid_ns 3. So it
sees the task as (1402,2), but needs to also store (1,3) and, on
restart, recreate a task with both those pids.

But I guess it will be pretty simple, and fall into place once we get
c/r semantics started.

thanks,
-serge

[original response]

I think that's the reasonable thing for people to do, but I don't think
we should force them to. I.e. there is no reason you shouldn't be able
to take one or two tasks out of a pidns and checkpoint them, and restart
them elsewhere. If it turns out they were talking to a third process
which wasn't checkpointed, well, too bad.

What you are more likely to need is a new clean set of namespaces to
restart in, but again I don't think we should enforce that. So whatever
mechanism we end up doing to implementing "clone_with_pid()", we should
handle -EBUSY correctly.

Anyway, why do you ask? (How does it follow from the conversation?)

I wasn't suggesting that it would be ok to only dump part of the pid
information, rather I was asking how we would do it correctly :)

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by serue on Fri, 25 May 2007 13:29:29 GMT

View Forum Message <> Reply to Message

Quoting Pavel Emelianov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > Quoting Eric W. Biederman (ebiederm@xmission.com):
> >> Pavel Emelianov <xemul@openvz.org> writes:
> >>
> >>> That's how OpenVZ sees the pid namespaces.
> >>>
> >>> The main idea is that kernel keeps operating with tasks pid
> >>> as it did before, but each task obtains one more pid for each
> >>> pid type - the virtual pid. When putting the pid to user or
> >>> getting the pid from it kernel operates with the virtual ones.
> >> Just a quick reaction.
> >>
> >> - I would very much like to see a minimum of 3 levels of pids,
> >>   being supported. Otherwise it is easy to overlook some of the
> >>   cases that are required to properly support nesting, which long
> >>   terms seems important.
> >
> > Pavel,
> >
> > If I wanted to start a virtual server and in there start some checkpoint
> > restart jobs, so I start a new pid namespace inside the c/r job, what
> > will happen?
>
> What will happen with this namespace on restore? What pids will
> you assign to it in the parent (but not that init) namespace?

No, no, my question is earlier. Maybe my use of the term
"checkpoint/restart job" is confusing, so let me call it a "batch job"
instead, with the understanding that it is started with the intent of
being safely checkpoint/restartable later on.

So in the original batch job, started in a vserver, what will the pids look like in the checkpoint/restart job?

But I think I know the answer - you'll leave vpid == pid for these
tasks, and only set vpid differently when restarting a job, since that's
when you really care?

So the only situation where there might be a shortcoming is when
restarting a job in a vserver?

-serge

> a. arbitrary: that means that you don't care that subgroup
>   of tasks in the VS namespace. Thus why don't move them
>   into separate namespace
> b. try to hold them as they were: this way is likely to fail
>   and can work w/o namespaces at all.
>
> So what's your answer?
>
> > a. second pidns unshare is refused
> > b. second pidns unshare is allowed, but c/r job is not visible
> > from the virtual server (but is from the global pidns)
> > c. second pidns unshare is allowed, and somehow the c/r job
> > is visible from the virtual server
> >
> > If (a), is this a short-term shortcoming for simplicity of prototype and
> > code review, or do you think it's actually the right thing t do long
> > term?
> >
> > thanks,
> > -serge
> >
> >> - Semantically fork is easier then unshare. Unshare can mean
> >>   a lot of things, and it is easy to pick a meaning that has weird
> >>   side effects. Your implementation has a serious problem in that you
> >>   change the value of getpid() at runtime. Glibc does not know how to
> >>   cope with the value of getpid() changing.
> >>
> >> Eric
> >

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by Pavel Emelianov on Fri, 25 May 2007 13:53:06 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelianov (xemul@openvz.org):

>> Serge E. Hallyn wrote:
>>> Quoting Eric W. Biederman (ebiederm@xmission.com):
>>>> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>>>>
>>>>>> 3. Cleaner logic for namespace migration: with this approach
>>>>>>   one need to save the virtual pid and let global one change;
>>>>>>   with Suka's logic this is not clear how to migrate the level
>>>>>>   2 namespace (concerning init to be level 0).
>>>>> This is a very good point.
>>>>>
>>>>> How *would* we migrate the pids at the second level?
>>>> As long as you don't try and restore pids into the initial pid namespace
>>>> it isn't a problem. You just record the pid hierarchy and the pid
>>>> for a task in that hierarchy. There really is nothing special going on
>>>> that should make migration hard.
>>>>
>>>> Or did I miss something?
>>> Hmm, no, i guess you are right. I was thinking that getting the pid for
>>> a process woudl be done purely from userspace, but I guess along with a
>>> kernel helper to *set* pids, we could also have a kernel helper to get
>>> all pids for all pid namespaces "above" that of the process doing the
>>> checkpoint.
>> So do you agree that if we migrate a VS we need to migrate the whole VS?
>
> I started to respond, then realized you were probably asking something
> different than I thought. My original response is below, but here is I
> think the answer to your question, which is important because I think
> your question might highlight a misunderstanding about the design of
> Suka's code.
>
> Let's say a vserver is started, and in there a pidns is started for a
> checkpoint/restart job. So let's say we have PID 13 in the root
> namespace starting PID 14 in a new namespace. So using (pid, pid_ns) as
> the terminology, we havd (13,1) as the parent process, and (14,1)=(1,2)
> as the init of the vserver. Let's ignore other tasks inthe vserver, and
> just talk about (1402,2) as the init of the checkpoint restart job, so
> it is (1402,2)=(1,3). And oh, yeah, (1402,2)=(1,3)=(2309,1).

Oh, this is heavy... Lets draw some diagrams.

You have a vserver with a namespace in it with a cpt job in it,
just like this:

```
[node. pids look like (N)]
 `- [vserver. pids look like (N,V)]
      `- [cpt job. pids look like (N,V,P)]
```

Is that OK?

We have task in "node" with pid (13) which spawns the task with
pid (14,1) into the "vserver", like this:

```
(13)
  `- (14,1)
```

If so, then what the notion (14,1)=(1,2) mean?

As far as the "cpt job" is concerned we have smth like this:

```
(13)
  `- (14,1)
        `- (1402,2,1)
```

where (1402,2,1) is the root of the "cpt job", right?

> Now when we want to migrate the vserver, a task in pid_ns 2 will look
> for all tasks with pids in pidns 2. That will automatically include all
> tasks in pid_ns 3. I think you thought I was asking how we would
> include pid_ns 3, and are asking whether it woudl be ok to not migrate
> pid_ns 3? (answer: it's irrelevant, all tasks in pid_ns 3 are also in
> pid_ns 2 - and in pid_ns 1).
>
> What I was actually asking was, in the same situation, how would the
> task in pid_ns 2 doing the checkpoint get the pids in pid_ns 3. So it
> sees the task as (1402,2), but needs to also store (1,3) and, on
> restart, recreate a task with both those pids.
>
> But I guess it will be pretty simple, and fall into place once we get
> c/r semantics started.
>
> thanks,
> -serge
>
> [original response]
>
> I think that's the reasonable thing for people to do, but I don't think
> we should force them to. I.e. there is no reason you shouldn't be able
> to take one or two tasks out of a pidns and checkpoint them, and restart
> them elsewhere. If it turns out they were talking to a third process
> which wasn't checkpointed, well, too bad.
>
> What you are more likely to need is a new clean set of namespaces to
> restart in, but again I don't think we should enforce that. So whatever
> mechanism we end up doing to implementing "clone_with_pid()", we should
> handle -EBUSY correctly.
>

> Anyway, why do you ask? (How does it follow from the conversation?)
>
> I wasn't suggesting that it would be ok to only dump part of the pid
> information, rather I was asking how we would do it correctly :)
>

Subject: Re: [PATCH 2/13] Small preparations for namespaces
Posted by serue on Fri, 25 May 2007 13:55:59 GMT

View Forum Message <> Reply to Message

Quoting Pavel Emelianov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > Quoting Pavel Emelianov (xemul@openvz.org):
> >> Serge E. Hallyn wrote:
> >>> Quoting Pavel Emelianov (xemul@openvz.org):
> >>>> This includes #ifdefs in get/put_pid_ns and rewriting
> >>>> the child_reaper() function to the more logical view.
> >>>>
> >>>> This doesn't fit logically into any other patch so
> >>>> I decided to make it separate.
> >>>>
> >>>> Signed-off-by: Pavel Emelianov <xemul@openvz.org>
> >>>>
> >>>> ---
> >>>>

```
> >>>> diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
> >>>> index 169c6c2..7af7191 100644
> >>>> --- a/include/linux/pid_namespace.h
> >>>> +++ b/include/linux/pid_namespace.h
> >>>> @@ -26,7 +26,9 @@ 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);
> >>>> @@ -34,12 +36,15 @@ 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 */
```
> >>> This can't be bisect-safe, right? You can't just use
> >>> tsk->nsproxy->pid_ns, as you've pointed out yourself.
> >> I can :) See - I have a proving BUG_ON() here.
> >
> > I didn't know BUG_ON()'s actually warded off bugs :)
>
> It does not, but it says to code reader that this call
> expects something special. In this case - tsk is expected
> to be current always. And it is.

I don't think that's sufficient.

It's been awhile so I'm fuzzy on the details, but I think we only fixed
the race by always returning init_pid_ns instead of tsk->nsproxy_pid_ns,
and tsk being current is not safe.

> > You've tested this with the infamous NFS testcase?
>
> What testcase do you mean?

http://lkml.org/lkml/2007/1/17/65

> > I don't see *why* it would work for you, but if you claim it does, I
> > guess you'd know better than I :)
>
> I don't get you here. I've checked that the task passed to
> child_reaper is current always. This BUG_ON prevents later
> code from passing arbitrary task to it.

I don't think that's enough.

thanks,
-serge

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by serue on Fri, 25 May 2007 14:25:50 GMT

View Forum Message <> Reply to Message

Quoting Pavel Emelianov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > Quoting Pavel Emelianov (xemul@openvz.org):
> >> Serge E. Hallyn wrote:
> >>> Quoting Eric W. Biederman (ebiederm@xmission.com):
> >>>> "Serge E. Hallyn" <serue@us.ibm.com> writes:
> >>>>
> >>>>>> 3. Cleaner logic for namespace migration: with this approach
> >>>>>>   one need to save the virtual pid and let global one change;
> >>>>>>   with Suka's logic this is not clear how to migrate the level
> >>>>>>   2 namespace (concerning init to be level 0).
> >>>>> This is a very good point.
> >>>>>
> >>>>> How *would* we migrate the pids at the second level?
> >>>> As long as you don't try and restore pids into the initial pid namespace
> >>>> it isn't a problem. You just record the pid hierarchy and the pid
> >>>> for a task in that hierarchy. There really is nothing special going on
> >>>> that should make migration hard.
> >>>>
> >>>> Or did I miss something?
> >>> Hmm, no, i guess you are right. I was thinking that getting the pid for
> >>> a process woudl be done purely from userspace, but I guess along with a
> >>> kernel helper to *set* pids, we could also have a kernel helper to get
> >>> all pids for all pid namespaces "above" that of the process doing the
> >>> checkpoint.
> >> So do you agree that if we migrate a VS we need to migrate the whole VS?
> >
> > I started to respond, then realized you were probably asking something
> > different than I thought. My original response is below, but here is I
> > think the answer to your question, which is important because I think
> > your question might highlight a misunderstanding about the design of
> > Suka's code.
> >
> > Let's say a vserver is started, and in there a pidns is started for a
> > checkpoint/restart job. So let's say we have PID 13 in the root
> > namespace starting PID 14 in a new namespace. So using (pid, pid_ns) as
> > the terminology, we havd (13,1) as the parent process, and (14,1)=(1,2)
> > as the init of the vserver. Let's ignore other tasks inthe vserver, and
> > just talk about (1402,2) as the init of the checkpoint restart job, so
> > it is (1402,2)=(1,3). And oh, yeah, (1402,2)=(1,3)=(2309,1).
>
> Oh, this is heavy... Lets draw some diagrams.
>
> You have a vserver with a namespace in it with a cpt job in it,
> just like this:
>
> [node. pids look like (N)]
>   `- [vserver. pids look like (N,V)]

```
>        `- [cpt job. pids look like (N,V,P)]
>
> Is that OK?
```

It's different from the notation I was using.

Let's stick to calling every process by a full "upid", i.e.
(pid, pid namespace #) because while it's longer it gives more
information.

```
> We have task in "node" with pid (13) which spawns the task with
> pid (14,1) into the "vserver", like this:
>
> (13)
>   `- (14,1)
>
> If so, then what the notion (14,1)=(1,2) mean?
```

It means that (pid 14, pid_ns 1) = (pid 1, pid_ns 2). It describes one
task, which in pid namespace 1 is known by pid 14, and in pid namespace
2 is known by pid 1.

(I see the repetative low numbers were confusing...)

```
> As far as the "cpt job" is concerned we have smth like this:
>
> (13)
>   `- (14,1)
>         `- (1402,2,1)
>
> where (1402,2,1) is the root of the "cpt job", right?
```

Sure, and in my notation this would be

```
  [(13,1)]
    `- [(14,1)(1,2)]
      `- [(2309,1)(1402,2)(1,3)]
```

Again each level is just one task, but known by several pids.

So coming back to the idea of checkpoint all of pid_ns=2, we would be
checkpointing both task [(14,1)(1,2)] and task [(2309,1)(1402,2)(1,3)].
And my question had been how would we access and store the fact that the
third task has pid (1,3), which we MUST store and reset, because that is
that task's active pid namespace, meaning it only knows itself as (1,3).

The task in pid namespace 2 which is doing the checkpointing generally
only knows the third task as (1402,2), so we need to provide a mechanism

for it to dump all pids in "higher" pid namespaces.

Note that, of course, pids in "lower" pid namespaces can be randomly
set. If we are restarting pid namespace 2 on a new system, it's
perfeclty ok for the pids to look like:

```
  [(467,1)]
    `- [(5597,1)(1,2)]
      `- [(5598,1)(1402,2)(1,3)]
```

Heh, or even

```
  [(14,1)(467,2)]
    `- [(444,1)(5597,2)(1,3)]
      `- [(445,1)(5598,2)(1402,3)(1,4)]
```

thanks,
-serge

> > Now when we want to migrate the vserver, a task in pid_ns 2 will look
> > for all tasks with pids in pidns 2. That will automatically include all
> > tasks in pid_ns 3. I think you thought I was asking how we would
> > include pid_ns 3, and are asking whether it woudl be ok to not migrate
> > pid_ns 3? (answer: it's irrelevant, all tasks in pid_ns 3 are also in
> > pid_ns 2 - and in pid_ns 1).
> >
> > What I was actually asking was, in the same situation, how would the
> > task in pid_ns 2 doing the checkpoint get the pids in pid_ns 3. So it
> > sees the task as (1402,2), but needs to also store (1,3) and, on
> > restart, recreate a task with both those pids.
> >
> > But I guess it will be pretty simple, and fall into place once we get
> > c/r semantics started.
> >
> > thanks,
> > -serge
> >
> > [original response]
> >
> > I think that's the reasonable thing for people to do, but I don't think
> > we should force them to. I.e. there is no reason you shouldn't be able
> > to take one or two tasks out of a pidns and checkpoint them, and restart
> > them elsewhere. If it turns out they were talking to a third process
> > which wasn't checkpointed, well, too bad.
> >
> > What you are more likely to need is a new clean set of namespaces to
> > restart in, but again I don't think we should enforce that. So whatever
> > mechanism we end up doing to implementing "clone_with_pid()", we should

> > handle -EBUSY correctly.
> >
> > Anyway, why do you ask? (How does it follow from the conversation?)
> >
> > I wasn't suggesting that it would be ok to only dump part of the pid
> > information, rather I was asking how we would do it correctly :)
> >

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by ebiederm on Fri, 25 May 2007 14:55:27 GMT
View Forum Message <> Reply to Message

Pavel Emelianov <xemul@openvz.org> writes:
>
> *I* would like to know how you migrate a *part* of a virtual
> server? What happens with pids, IPC ids, network connections?
>
> There are many entities in VS that are not bound to task, but to
> VS and if you migrate only half of them you're risking in loosing
> the integrity of the VS. If you don't care it - why do you need
> namespaces at all?

Well there are other uses for namespaces like providing a context
for private mounts.

That said the concept for migration is not a partial VS. But
a nested VS.

Eric

Subject: Re: [PATCH 11/13] Changes to show virtual ids to user
Posted by ebiederm on Fri, 25 May 2007 15:48:29 GMT
View Forum Message <> Reply to Message

Pavel Emelianov <xemul@openvz.org> writes:

> Eric W. Biederman wrote:
>> Pavel Emelianov <xemul@sw.ru> writes:
>>
>>> That's true. Sending of signal from parent ns to children
>>> is tricky question. It has many solutions, I wanted to
>>> discuss which one is better:
>>
>> With unix domain sockets and the like it is conceivable we get
>> a pid transfer from one namespace to another and both namespaces

>> are leaf namespaces. I don't remember we can get a leaf to leaf
>> transfer when sending signals.
>
> We should not allow any transfer from leaf NS to leaf NS.
> Should I explain why?

In a checkpointable context it is a bad thing, and we can prevent it
by carefully setting up all of the namespaces.

However it is a fundamental possibility that exists, and because we
can avoid it with careful setup. I don't see a reason to deny it
if something was either inadvertantly or explicitly causes it
to happen.

Do you have another reason for denying the transfer that I'm
not thinking of?

>>
>> The worst case I can see with pid == 0. Is that it would be a bug
>> that we can fix later. For other cases it would seem to be a user
>> space API thing that we get stuck with for all time.
>
> We cannot trust userspace application to expect some pid other than
> positive. All that we can is either use some always-absent pid or
> send the signal as SI_KERNEL.
>
> Our experience show that making decisions like above causes random
> applications failures that are hard (or even impossible) to debug.

Ok. So I guess I see what you are proposing is picking an arbitrary
pid, say pid == 2, and reserving that in all pid namespaces and using
it when we have a pid that does not map to a specific namespace. I'm
fine with that.

All I care about is that we have a solution, preferably simple,
to the non-mapped pid problem.

Eric

Subject: Re: [PATCH 5/13] Expand the pid/task seeking functions set
Posted by Sukadev Bhattiprolu on Fri, 25 May 2007 23:36:58 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@openvz.org] wrote:
| Dave Hansen wrote:
| > On Thu, 2007-05-24 at 16:50 +0400, Pavel Emelianov wrote:

```
| >> +struct pid * fastcall __find_vpid(int nr, struct pid_namespace *ns)
| >> +{
| >> +#ifdef CONFIG_PID_NS
| >> +	struct hlist_node *elem;
| >> +	struct pid *pid;
| >> +#endif
| >> +
| >> +	if (ns == &init_pid_ns)
| >> +		return find_pid(nr);
| >> +
| >> +#ifdef CONFIG_PID_NS
| >> +	hlist_for_each_entry_rcu(pid, elem,
| >> +			&vpid_hash[vpid_hashfn(nr, ns)], vpid_chain) {
| >> +		if (pid->vnr == nr && pid->ns == ns)
| >> +			return pid;
| >> +	}
| >> +#endif
| >> +	return NULL;
| >> +}
| >
```
| > I am a bit worried that there are too many #ifdefs here. Your patch
| > series adds ~20 of them, and they look to me to be mostly in .c files.
| > Section 2 in SubmittingPatches somewhat discourages this.
| >
| > Do you have any plans for cleaning these up?
|
| Sure I have. But this approach makes review simpler - everyone
| explicitly see what exact actions are taken in each place. In
| the second iteration this will be make in a more elegant way
| like making static inline stubs etc.
|
| This set is a kind of RFC and proof-of-concept. I didn't intent
| this to be merged to any tree as is. That's why a attached the
| lats patch with strut in proc to observe the whole tree.
|
| BTW, question to Sukadev - how did you test your patches? I do
| know that ps utility doesn't work without full /proc tree and
| I don's see similar hacks in your patchset.

Patches (#13 and #14) in my patchset allow remounting /proc in
a child namespace. So the script (lxc-wrap.sh Patch-0) remounts
/proc when it enters the new namespace. "ps -e" in the namespace
only shows processes from that namespace.

"ps -e" in init pid ns shows processes in child namespaces with
numeric pids (pid_t) from init pid ns.

|
| > -- Dave
| >
| >

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by Pavel Emelianov on Mon, 28 May 2007 07:48:20 GMT

View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelianov (xemul@openvz.org):
>> Serge E. Hallyn wrote:
>>> Quoting Pavel Emelianov (xemul@openvz.org):
>>>> Serge E. Hallyn wrote:
>>>>> Quoting Eric W. Biederman (ebiederm@xmission.com):
>>>>>> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>>>>>>
>>>>>>>>> 3. Cleaner logic for namespace migration: with this approach
>>>>>>>>>   one need to save the virtual pid and let global one change;
>>>>>>>>>   with Suka's logic this is not clear how to migrate the level
>>>>>>>>>   2 namespace (concerning init to be level 0).
>>>>>>> This is a very good point.
>>>>>>>
>>>>>>> How *would* we migrate the pids at the second level?
>>>>>> As long as you don't try and restore pids into the initial pid namespace
>>>>>> it isn't a problem. You just record the pid hierarchy and the pid
>>>>>> for a task in that hierarchy. There really is nothing special going on
>>>>>> that should make migration hard.
>>>>>>
>>>>>> Or did I miss something?
>>>>> Hmm, no, i guess you are right. I was thinking that getting the pid for
>>>>> a process woudl be done purely from userspace, but I guess along with a
>>>>> kernel helper to *set* pids, we could also have a kernel helper to get
>>>>> all pids for all pid namespaces "above" that of the process doing the
>>>>> checkpoint.
>>>> So do you agree that if we migrate a VS we need to migrate the whole VS?
>>> I started to respond, then realized you were probably asking something
>>> different than I thought. My original response is below, but here is I
>>> think the answer to your question, which is important because I think
>>> your question might highlight a misunderstanding about the design of
>>> Suka's code.
>>>
>>> Let's say a vserver is started, and in there a pidns is started for a

>>> checkpoint/restart job. So let's say we have PID 13 in the root
>>> namespace starting PID 14 in a new namespace. So using (pid, pid_ns) as
>>> the terminology, we havd (13,1) as the parent process, and (14,1)=(1,2)
>>> as the init of the vserver. Let's ignore other tasks inthe vserver, and
>>> just talk about (1402,2) as the init of the checkpoint restart job, so
>>> it is (1402,2)=(1,3). And oh, yeah, (1402,2)=(1,3)=(2309,1).
>> Oh, this is heavy... Lets draw some diagrams.
>>
>> You have a vserver with a namespace in it with a cpt job in it,
>> just like this:
>>
>> [node. pids look like (N)]
>>   `- [vserver. pids look like (N,V)]
>>   `- [cpt job. pids look like (N,V,P)]
>>
>> Is that OK?
>
> It's different from the notation I was using.
>
> Let's stick to calling every process by a full "upid", i.e.
> (pid, pid namespace #) because while it's longer it gives more
> information.
>
>> We have task in "node" with pid (13) which spawns the task with
>> pid (14,1) into the "vserver", like this:
>>
>> (13)
>>   `- (14,1)
>>
>> If so, then what the notion (14,1)=(1,2) mean?
>
> It means that (pid 14, pid_ns 1) = (pid 1, pid_ns 2). It describes one
> task, which in pid namespace 1 is known by pid 14, and in pid namespace
> 2 is known by pid 1.
>
> (I see the repetative low numbers were confusing...)
>
>> As far as the "cpt job" is concerned we have smth like this:
>>
>> (13)
>>   `- (14,1)
>>   `- (1402,2,1)
>>
>> where (1402,2,1) is the root of the "cpt job", right?
>
> Sure, and in my notation this would be
>
>   [(13,1)]

```
>       `- [(14,1)(1,2)]
>          `- [(2309,1)(1402,2)(1,3)]
>
```
> Again each level is just one task, but known by several pids.
>
> So coming back to the idea of checkpoint all of pid_ns=2, we would be
> checkpointing both task [(14,1)(1,2)] and task [(2309,1)(1402,2)(1,3)].
> And my question had been how would we access and store the fact that the
> third task has pid (1,3), which we MUST store and reset, because that is
> that task's active pid namespace, meaning it only knows itself as (1,3).
>
> The task in pid namespace 2 which is doing the checkpointing generally
> only knows the third task as (1402,2), so we need to provide a mechanism
> for it to dump all pids in "higher" pid namespaces.
>
> Note that, of course, pids in "lower" pid namespaces can be randomly
> set. If we are restarting pid namespace 2 on a new system, it's
> perfeclty ok for the pids to look like:
>
```
>    [(467,1)]
>       `- [(5597,1)(1,2)]
>          `- [(5598,1)(1402,2)(1,3)]
>
```
> Heh, or even
>
```
>    [(14,1)(467,2)]
>       `- [(444,1)(5597,2)(1,3)]
>          `- [(445,1)(5598,2)(1402,3)(1,4)]
```

Hmm. I see. So you don't care that the pids in the namespace #2 are still
the same. I can understand that politics for namespace #1, but for #2...

OK, if you need this let us go on with such model, but I'd like to see
the CONFIG_PID_NS_MULTILEVEL for this. Or at least CONFIG_PID_NS_FLAT for
my model as we do not need to sacrifice the performance to such generic
behavior.

Thanks,
Pavel.

>
> thanks,
> -serge
>
>>> Now when we want to migrate the vserver, a task in pid_ns 2 will look
>>> for all tasks with pids in pidns 2. That will automatically include all
>>> tasks in pid_ns 3. I think you thought I was asking how we would
>>> include pid_ns 3, and are asking whether it woudl be ok to not migrate

>>> pid_ns 3? (answer: it's irrelevant, all tasks in pid_ns 3 are also in
>>> pid_ns 2 - and in pid_ns 1).
>>>
>>> What I was actually asking was, in the same situation, how would the
>>> task in pid_ns 2 doing the checkpoint get the pids in pid_ns 3. So it
>>> sees the task as (1402,2), but needs to also store (1,3) and, on
>>> restart, recreate a task with both those pids.
>>>
>>> But I guess it will be pretty simple, and fall into place once we get
>>> c/r semantics started.
>>>
>>> thanks,
>>> -serge
>>>
>>> [original response]
>>>
>>> I think that's the reasonable thing for people to do, but I don't think
>>> we should force them to. I.e. there is no reason you shouldn't be able
>>> to take one or two tasks out of a pidns and checkpoint them, and restart
>>> them elsewhere. If it turns out they were talking to a third process
>>> which wasn't checkpointed, well, too bad.
>>>
>>> What you are more likely to need is a new clean set of namespaces to
>>> restart in, but again I don't think we should enforce that. So whatever
>>> mechanism we end up doing to implementing "clone_with_pid()", we should
>>> handle -EBUSY correctly.
>>>
>>> Anyway, why do you ask? (How does it follow from the conversation?)
>>>
>>> I wasn't suggesting that it would be ok to only dump part of the pid
>>> information, rather I was asking how we would do it correctly :)
>>>
>

Subject: Re: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by dev on Mon, 28 May 2007 11:50:02 GMT

View Forum Message <> Reply to Message

Pavel Emelianov wrote:

>>> Though I haven't
>>>seen any problems with glibc for many years running OpenVZ and I think,
>>>that if glibc will want to cache this getpid() value we can teach it to
>>>uncache this value in case someone called unshare() with CLONE_NEWPIDS.
>>
>>glibc very much caches the results of getpid().
>

>
> Can you prove it? We have run OpenVZ for many years and with many
> userspace configurations and we haven't seen the problems with
> glibc ever.

Pavel, but we always do full namespace entering with fork()'s and such
actions. So we simply couldn't trigger getpid() caching.

Thanks,
Kirill

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by ebiederm on Tue, 29 May 2007 04:30:45 GMT
View Forum Message <> Reply to Message

Pavel Emelianov <xemul@openvz.org> writes:

> Hmm. I see. So you don't care that the pids in the namespace #2 are still
> the same. I can understand that politics for namespace #1, but for #2...

I'm confused, I think the statement above is wrong.

If we just checkpoint/restart a leaf pid namespace we don't care about
the other pids, in other namespace.

If we checkpoint/restart a pid namespace with another pid namespace
nested inside it we need to preserve the pids in the pid namespace we
are checkpointing and in a nested pid namespaces.

Pids in namespaces that none of the process we are migrating cannot
see we do not care about. (i.e. the init pid namespace, and possibly
some of it's children)

> OK, if you need this let us go on with such model, but I'd like to see
> the CONFIG_PID_NS_MULTILEVEL for this. Or at least CONFIG_PID_NS_FLAT for
> my model as we do not need to sacrifice the performance to such generic
> behavior.

Where is the world would a performance sacrafice come in? If you
happen to be using a deeply nested pid namespace I can see a small
performance hit, there is fundamentally more to do. However if you
don't use a nested pid namespace there should not be more work todo
and it should be impossible to measure the over head.

Further 3 levels should be as simple to implement and as cheap as two
levels. Because we can continue to use static allocation.

Eric

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by Pavel Emelianov on Tue, 29 May 2007 07:47:35 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Pavel Emelianov <xemul@openvz.org> writes:
>
>> Hmm. I see. So you don't care that the pids in the namespace #2 are still
>> the same. I can understand that politics for namespace #1, but for #2...
>
> I'm confused, I think the statement above is wrong.
>
> If we just checkpoint/restart a leaf pid namespace we don't care about
> the other pids, in other namespace.
>
> If we checkpoint/restart a pid namespace with another pid namespace
> nested inside it we need to preserve the pids in the pid namespace we
> are checkpointing and in a nested pid namespaces.
>
> Pids in namespaces that none of the process we are migrating cannot
> see we do not care about. (i.e. the init pid namespace, and possibly
> some of it's children)
>
>> OK, if you need this let us go on with such model, but I'd like to see
>> the CONFIG_PID_NS_MULTILEVEL for this. Or at least CONFIG_PID_NS_FLAT for
>> my model as we do not need to sacrifice the performance to such generic
>> behavior.
>
> Where is the world would a performance sacrafice come in? If you

Easy! Consider the problem of getting a list of pids for proc. In case
of flat layout we just take a number from a known structure. In case of
nested pids we have to scan through the list of pid_elem-s or lookup
the hash or something similar.

The same stays true for wait() when we have to compare pids in the
eligible_child(), for setpgid(), terminal ioctls and so on and so forth.

Not to be unfounded I will measure booth cases with unixbench's spawn,
execl and shell tests and with "ps -xaf" and report the results. All will
be run in init namespace and in "level one" namespace. If the flat layout
wins (with noticeable difference) I would insist having two of them. Agree?

> happen to be using a deeply nested pid namespace I can see a small
> performance hit, there is fundamentally more to do. However if you

> don't use a nested pid namespace there should not be more work todo
> and it should be impossible to measure the over head.
>
> Further 3 levels should be as simple to implement and as cheap as two
> levels. Because we can continue to use static allocation.

Wait a bit. Do you mean that there's enough to have only 3 levels of
namespaces? I.e. to have a struct pid look like

```
struct pid {
 int pid;
 int pid1; /* for first level */
 int pid2; /* for 2nd level */
 ...
}
```

?

> Eric
>

Subject: Re: [PATCH 11/13] Changes to show virtual ids to user
Posted by Cedric Le Goater on Tue, 29 May 2007 12:32:15 GMT
View Forum Message <> Reply to Message

Hello !

>>> The worst case I can see with pid == 0. Is that it would be a bug
>>> that we can fix later. For other cases it would seem to be a user
>>> space API thing that we get stuck with for all time.
>> We cannot trust userspace application to expect some pid other than
>> positive. All that we can is either use some always-absent pid or
>> send the signal as SI_KERNEL.
>>
>> Our experience show that making decisions like above causes random
<>> applications failures that are hard (or even impossible) to debug.

> Ok. So I guess I see what you are proposing is picking an arbitrary
> pid, say pid == 2, and reserving that in all pid namespaces and using
> it when we have a pid that does not map to a specific namespace. I'm
> fine with that.
>
> All I care about is that we have a solution, preferably simple,
> to the non-mapped pid problem.

Pavel, are you against using pid == 0 and setting si_code to SI_KERNEL ?

C.

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by ebiederm on Tue, 29 May 2007 12:36:06 GMT
View Forum Message <> Reply to Message

Pavel Emelianov <xemul@openvz.org> writes:

> Eric W. Biederman wrote:
>> Pavel Emelianov <xemul@openvz.org> writes:
>> Where is the world would a performance sacrafice come in? If you
>
> Easy! Consider the problem of getting a list of pids for proc. In case
> of flat layout we just take a number from a known structure. In case of
> nested pids we have to scan through the list of pid_elem-s or lookup
> the hash or something similar.

We walk through the pidmap. That should not change either way.

I'm actually not horribly fond of walking through the pidmap but
it was needed for correctness so we could have a stable token we could
return to user space for restarting readdir in /proc.

> The same stays true for wait() when we have to compare pids in the
> eligible_child(), for setpgid(), terminal ioctls and so on and so forth.

We should be comparing struct pid pointers not user space pid_t values.
With that being the case we should convert at the edge of user space
and all should be good.

>> happen to be using a deeply nested pid namespace I can see a small
>> performance hit, there is fundamentally more to do. However if you
>> don't use a nested pid namespace there should not be more work todo
>> and it should be impossible to measure the over head.
>>
>> Further 3 levels should be as simple to implement and as cheap as two
>> levels. Because we can continue to use static allocation.
>
> Wait a bit. Do you mean that there's enough to have only 3 levels of
> namespaces? I.e. to have a struct pid look like
> struct pid {
> int pid;
> int pid1; /* for first level */
> int pid2; /* for 2nd level */
> ...
> }
> ?

Initially yes. 3 levels should be enough. Ultimately we may want more
but that should be a small tweak at the implementation level. Nothing
outside of the pid functions should care.

Eric

Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by ebiederm on Tue, 29 May 2007 13:07:13 GMT
View Forum Message <> Reply to Message

Hmm. I seem to have forgotten to send this one.

Pavel Emelianov <xemul@openvz.org> writes:

> Eric W. Biederman wrote:

> Generic structures are not always needed. Say, why don't we
> have N-level page tables in kernel? Why not make them generic?
> What if some ia128 architecture will require 7-level tables!?

PID namespaces unlike the other namespaces are fundamentally nested.
Which is an unfortunate pain. But if you want to allow nesting
of containers of different types such as system containers and
application containers you need nested PID namespaces.

>> Having more then two layers means we are prepared to use pid namespaces more
>> generally. It really isn't that much harder.
>
> It is not, but do we need to spend so much time on solving
> not relevant problems?

It is relevant to some of us. Therefore it is a relevant problem.

>>>> - Semantically fork is easier then unshare. Unshare can mean
>>> This is not. When you fork, the kid shares the session and the
>>> group with its parent, but moving this pids to new ns is bad - the
>>> parent will happen to be half-moved. Thus you need to break the
>>> session and the group in fork(), but this is extra complexity.
>>
>> Nope. You will just need to have the child call setsid() if
>> you don't want to share the session and the group.
>
> Of course, but setsid() must be done *before* creating a new
> namespace, Otherwise you will have a half-inserted into new
> namespace task. This sounds awful.

We can experience weird interactions, but not really worse then the
sending a signal from outside the namespace. So we may want to
map the pids of the session and the pgrp into the new namespace but
functionally it's not really a big deal, and we can call setsid

after the fork.

>>>>   a lot of things, and it is easy to pick a meaning that has weird
>>>>   side effects. Your implementation has a serious problem in that you
>>>>   change the value of getpid() at runtime. Glibc does not know how to
>>>>   cope with the value of getpid() changing.
>>> This pid changing happens only once per task lifetime.
>>
>> Unshare isn't once per task lifetime, unless you added some extra
>> constraints.
>
> It is once. You create a new namespace and that's all.

What prevents you from calling unshare multiple times?

>>> Though I haven't
>>> seen any problems with glibc for many years running OpenVZ and I think,
>>> that if glibc will want to cache this getpid() value we can teach it to
>>> uncache this value in case someone called unshare() with CLONE_NEWPIDS.
>>
>> glibc very much caches the results of getpid().
>
> Can you prove it? We have run OpenVZ for many years and with many
> userspace configurations and we haven't seen the problems with
> glibc ever.

Yes. I did a migration prototype in user space. Migrated a process
to a new pid, but getpid returned the pid before migration. So I
investigated why, including reading the glibc code. glibc cache the
pid value. Once the value is cached only a fork invalidates the
cache.

From: nptl/sysdeps/unix/sysv/linux/getpid.c

```
pid_t
__getpid (void)
{
  pid_t result = THREAD_GETMEM (THREAD_SELF, pid);
  if (__builtin_expect (result <= 0, 0))
    result = really_getpid (result);
  return result;
}
```

THREAD_GETMEM is a memory read.
really_getpid is the syscall.

Eric

Subject: Re: [PATCH 11/13] Changes to show virtual ids to user
Posted by Pavel Emelianov on Thu, 31 May 2007 07:57:37 GMT
View Forum Message <> Reply to Message

Cedric Le Goater wrote:
> Hello !
>
>>>> The worst case I can see with pid == 0. Is that it would be a bug
>>>> that we can fix later. For other cases it would seem to be a user
>>>> space API thing that we get stuck with for all time.
>>> We cannot trust userspace application to expect some pid other than
>>> positive. All that we can is either use some always-absent pid or
>>> send the signal as SI_KERNEL.
>>>
>>> Our experience show that making decisions like above causes random
> <>> applications failures that are hard (or even impossible) to debug.
>
>> Ok. So I guess I see what you are proposing is picking an arbitrary
>> pid, say pid == 2, and reserving that in all pid namespaces and using
>> it when we have a pid that does not map to a specific namespace. I'm
>> fine with that.
>>
>> All I care about is that we have a solution, preferably simple,
>> to the non-mapped pid problem.
>
> Pavel, are you against using pid == 0 and setting si_code to SI_KERNEL ?

I think I am. A quick grep through the code revealed one place where
this can happen, so I believe application are (have to be) somehow
prepared to this.

> C.
>

Subject: Re: [PATCH 11/13] Changes to show virtual ids to user
Posted by Pavel Emelianov on Thu, 31 May 2007 08:00:17 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Cedric Le Goater wrote:
>> Hello !
>>
>>>>> The worst case I can see with pid == 0. Is that it would be a bug
>>>>> that we can fix later. For other cases it would seem to be a user
>>>>> space API thing that we get stuck with for all time.
>>>> We cannot trust userspace application to expect some pid other than
>>>> positive. All that we can is either use some always-absent pid or

>>>> send the signal as SI_KERNEL.
>>>>
>>>> Our experience show that making decisions like above causes random
>> <>> applications failures that are hard (or even impossible) to debug.
>>
>>> Ok. So I guess I see what you are proposing is picking an arbitrary
>>> pid, say pid == 2, and reserving that in all pid namespaces and using
>>> it when we have a pid that does not map to a specific namespace. I'm
>>> fine with that.
>>>
>>> All I care about is that we have a solution, preferably simple,
>>> to the non-mapped pid problem.
>> Pavel, are you against using pid == 0 and setting si_code to SI_KERNEL ?
>
> I think I am. A quick grep through the code revealed one place where

Sorry. I have misprinted. I meant "I think I am *NOT*". My bad :(

> this can happen, so I believe application are (have to be) somehow
> prepared to this.
>
>> C.
>>
>
>

Subject: Re: [PATCH 11/13] Changes to show virtual ids to user
Posted by ebiederm on Thu, 31 May 2007 11:26:12 GMT
View Forum Message <> Reply to Message

Pavel Emelianov <xemul@openvz.org> writes:

> Pavel Emelianov wrote:
>> Cedric Le Goater wrote:
>>> Hello !
>>>
>>>>>> The worst case I can see with pid == 0. Is that it would be a bug
>>>>>> that we can fix later. For other cases it would seem to be a user
>>>>>> space API thing that we get stuck with for all time.
>>>>> We cannot trust userspace application to expect some pid other than
>>>>> positive. All that we can is either use some always-absent pid or
>>>>> send the signal as SI_KERNEL.
>>>>>
>>>>> Our experience show that making decisions like above causes random
>>> <>> applications failures that are hard (or even impossible) to debug.
>>>
>>>> Ok. So I guess I see what you are proposing is picking an arbitrary

>>>> pid, say pid == 2, and reserving that in all pid namespaces and using
>>>> it when we have a pid that does not map to a specific namespace. I'm
>>>> fine with that.
>>>>
>>>> All I care about is that we have a solution, preferably simple,
>>>> to the non-mapped pid problem.
>>> Pavel, are you against using pid == 0 and setting si_code to SI_KERNEL ?
>>
>> I think I am. A quick grep through the code revealed one place where
>
> Sorry. I have misprinted. I meant "I think I am *NOT*". My bad :(
>
>> this can happen, so I believe application are (have to be) somehow
>> prepared to this.

Where was this. I'd like to follow your complete line of thinking.

Eric

Subject: Re: [PATCH 11/13] Changes to show virtual ids to user
Posted by Pavel Emelianov on Thu, 31 May 2007 11:46:01 GMT

View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Pavel Emelianov <xemul@openvz.org> writes:
>
>> Pavel Emelianov wrote:
>>> Cedric Le Goater wrote:
>>>> Hello !
>>>>
>>>>>>> The worst case I can see with pid == 0. Is that it would be a bug
>>>>>>> that we can fix later. For other cases it would seem to be a user
>>>>>>> space API thing that we get stuck with for all time.
>>>>>> We cannot trust userspace application to expect some pid other than
>>>>>> positive. All that we can is either use some always-absent pid or
>>>>>> send the signal as SI_KERNEL.
>>>>>>
>>>>>> Our experience show that making decisions like above causes random
>>>> <>> applications failures that are hard (or even impossible) to debug.
>>>>
>>>>> Ok. So I guess I see what you are proposing is picking an arbitrary
>>>>> pid, say pid == 2, and reserving that in all pid namespaces and using
>>>>> it when we have a pid that does not map to a specific namespace. I'm
>>>>> fine with that.
>>>>>
>>>>> All I care about is that we have a solution, preferably simple,
>>>>> to the non-mapped pid problem.

>>>> Pavel, are you against using pid == 0 and setting si_code to SI_KERNEL ?
>>> I think I am. A quick grep through the code revealed one place where
>> Sorry. I have misprinted. I meant "I think I am *NOT*". My bad :(
>>
>>> this can happen, so I believe application are (have to be) somehow
>>> prepared to this.
>
> Where was this. I'd like to follow your complete line of thinking.

The line concerning why I think that sending a signal from
SI_KERNEL is good solution?

> Eric
>

Subject: Re: [PATCH 11/13] Changes to show virtual ids to user
Posted by ebiederm on Thu, 31 May 2007 13:41:22 GMT
View Forum Message <> Reply to Message

Pavel Emelianov <xemul@openvz.org> writes:

> Eric W. Biederman wrote:
>> Pavel Emelianov <xemul@openvz.org> writes:
>>
>>> Pavel Emelianov wrote:
>>>> Cedric Le Goater wrote:
>>>>> Hello !
>>>>>
>>>>>>>> The worst case I can see with pid == 0. Is that it would be a bug
>>>>>>>> that we can fix later. For other cases it would seem to be a user
>>>>>>>> space API thing that we get stuck with for all time.
>>>>>>> We cannot trust userspace application to expect some pid other than
>>>>>>> positive. All that we can is either use some always-absent pid or
>>>>>>> send the signal as SI_KERNEL.
>>>>>>>
>>>>>>> Our experience show that making decisions like above causes random
>>>>> <>> applications failures that are hard (or even impossible) to debug.
>>>>>
>>>>>> Ok. So I guess I see what you are proposing is picking an arbitrary
>>>>>> pid, say pid == 2, and reserving that in all pid namespaces and using
>>>>>> it when we have a pid that does not map to a specific namespace. I'm
>>>>>> fine with that.
>>>>>>
>>>>>> All I care about is that we have a solution, preferably simple,
>>>>>> to the non-mapped pid problem.
>>>>> Pavel, are you against using pid == 0 and setting si_code to SI_KERNEL ?
>>>> I think I am. A quick grep through the code revealed one place where

>>> Sorry. I have misprinted. I meant "I think I am *NOT*". My bad :(
>>>
>>>> this can happen, so I believe application are (have to be) somehow
>>>> prepared to this.
>>
>> Where was this. I'd like to follow your complete line of thinking.
>
> The line concerning why I think that sending a signal from
> SI_KERNEL is good solution?

Let me just restate everything to be certain we are not getting
confused.

The problem was what to do with signals from unmmaped pids.

You have just said pid == 0 with SI_KERNEL seems to work.

The kernel occasionally sends signal that way already.

The primary argument against this in my memory was that we
a user space application might treat the kernel case special
(more trust), so it might be a bad idea.

I believe what you just said was that user space has to be ready
to handle signals from pid == 0 with SI_KERNEL set. Therefore this
should just work. I don't think you have addressed the levels of
trust in user space issue or I might be confused.

Eric