Subject: [RFC][PATCH] VPIDs: Virtualization of PIDs (OpenVZ approach)
Posted by Kirill Korotaev on Thu, 02 Feb 2006 15:54:21 GMT
View Forum Message <> Reply to Message

[RFC][PATCH] VPIDs: Virtualization of PIDs (OpenVZ approach)

OpenVZ project would like to present another approach for VPIDs
developed by Alexey Kuznetsov. These patches were heavily tested and
used for half a year in OpenVZ already.

OpenVZ VPIDs serves the same purpose of VPS (container) checkpointing
and restore. OpenVZ VPS checkpointing code will be released soon.

These VPIDs patches look less intrusive than IBM's, so I would ask to
consider it for inclusion. Other projects can benefit from this code as
well.

Patch set consists of 7 patches.
The main patches are:
- diff-vpids-macro, which introduces pid access interface
- diff-vpids-core, which introduces pid translations where required
using new interfaces
- diff-vpids-virt, which introduces virtual pids handling in pid.c and
pid mappings for VPSs

Patches are against latest 2.6.16-rc1-git6

Kirill Korotaev,
OpenVZ team

Subject: [RFC][PATCH 1/7] VPIDs: add VPID config option
Posted by dev on Thu, 02 Feb 2006 16:16:20 GMT
View Forum Message <> Reply to Message

Add CONFIG_VIRTUAL_PIDS config option. Too simple one.

Kirill

```
--- ./init/Kconfig.vpid_conf 2006-02-02 14:15:35.145785768 +0300
+++ ./init/Kconfig 2006-02-02 14:31:22.904704512 +0300
@@ -316,6 +316,15 @@ config PRINTK
    very difficult to diagnose system problems, saying N here is
    strongly discouraged.

+config VIRTUAL_PIDS
+ default n
+ bool "Enable virtual pids support"
```

```
+ help
+   This option turns on process' ids virtualisation. Each
+   task has two types of pids assigned - system and virtual.
+   The latter is the one visible to user-space. When off
+   virtual pid is always equal to system one.
+
 config BUG
  bool "BUG() support" if EMBEDDED
  default y
```

Subject: [RFC][PATCH 2/7] VPIDs: pid/vpid conversions
Posted by dev on Thu, 02 Feb 2006 16:21:13 GMT
View Forum Message <> Reply to Message

This is one of the major patches,
it adds vpid-to-pid conversions by placing macros
introduced in diff-vpid-macro patch.

Note that in CONFIG_VIRTUAL_PIDS=n case these macros expand to default code.

Kirill

```
--- ./drivers/char/tty_io.c.vpid_core 2006-02-02 14:15:33.271070768 +0300
+++ ./drivers/char/tty_io.c 2006-02-02 14:33:58.795005584 +0300
@@ -2366,7 +2366,7 @@ static int tiocgpgrp(struct tty_struct *
   */
  if (tty == real_tty && current->signal->tty != real_tty)
   return -ENOTTY;
- return put_user(real_tty->pgrp, p);
+ return put_user(pid_type_to_vpid(PIDTYPE_PGID, real_tty->pgrp), p);
 }

 static int tiocspgrp(struct tty_struct *tty, struct tty_struct *real_tty, pid_t __user *p)
@@ -2386,6 +2386,9 @@ static int tiocspgrp(struct tty_struct *
   return -EFAULT;
  if (pgrp < 0)
   return -EINVAL;
+ pgrp = vpid_to_pid(pgrp);
+ if (pgrp < 0)
+  return -EPERM;
  if (session_of_pgrp(pgrp) != current->signal->session)
   return -EPERM;
  real_tty->pgrp = pgrp;
@@ -2402,7 +2405,7 @@ static int tiocgsid(struct tty_struct *t
   return -ENOTTY;
  if (real_tty->session <= 0)
   return -ENOTTY;
```

```
- return put_user(real_tty->session, p);
+ return put_user(pid_type_to_vpid(PIDTYPE_SID, real_tty->session), p);
 }

 static int tiocsetd(struct tty_struct *tty, int __user *p)
--- ./fs/binfmt_elf.c.vpid_core 2006-02-02 14:15:34.478887152 +0300
+++ ./fs/binfmt_elf.c 2006-02-02 14:33:58.797005280 +0300
@@ -1276,10 +1276,10 @@ 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 = process_group(p);
- prstatus->pr_sid = p->signal->session;
+ prstatus->pr_pid = virt_pid(p);
+ prstatus->pr_ppid = virt_pid(p->parent);
+ prstatus->pr_pgrp = virt_pgid(p);
+ prstatus->pr_sid = virt_sid(p);
  if (thread_group_leader(p)) {
   /*
    * This is the record for the group leader.  Add in the
@@ -1322,10 +1322,10 @@ 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 = process_group(p);
- psinfo->pr_sid = p->signal->session;
+ psinfo->pr_pid = virt_pid(p);
+ psinfo->pr_ppid = virt_pid(p->parent);
+ psinfo->pr_pgrp = virt_pgid(p);
+ psinfo->pr_sid = virt_sid(p);

  i = p->state ? ffz(~p->state) + 1 : 0;
  psinfo->pr_state = i;
--- ./fs/exec.c.vpid_core 2006-02-02 14:15:55.352713848 +0300
+++ ./fs/exec.c 2006-02-02 14:33:58.799004976 +0300
@@ -1282,7 +1282,7 @@ static void format_corename(char *corena
    case 'p':
     pid_in_pattern = 1;
     rc = snprintf(out_ptr, out_end - out_ptr,
-          "%d", current->tgid);
+          "%d", virt_tgid(current));
     if (rc > out_end - out_ptr)
      goto out;
     out_ptr += rc;
```

```
@@ -1354,7 +1354,7 @@ static void format_corename(char *corena
  if (!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", virt_tgid(current));
   if (rc > out_end - out_ptr)
    goto out;
   out_ptr += rc;
--- ./fs/fcntl.c.vpid_core 2006-02-02 14:15:34.521880616 +0300
+++ ./fs/fcntl.c 2006-02-02 14:33:58.800004824 +0300
@@ -251,6 +251,7 @@ static int setfl(int fd, struct file * f
 static void f_modown(struct file *filp, unsigned long pid,
                      uid_t uid, uid_t euid, int force)
 {
+ pid = comb_vpid_to_pid(pid);
  write_lock_irq(&filp->f_owner.lock);
  if (force || !filp->f_owner.pid) {
   filp->f_owner.pid = pid;
@@ -317,7 +318,7 @@ static long do_fcntl(int fd, unsigned in
    * current syscall conventions, the only way
    * to fix this will be in libc.
    */
-  err = filp->f_owner.pid;
+  err = comb_pid_to_vpid(filp->f_owner.pid);
   force_successful_syscall_return();
   break;
  case F_SETOWN:
--- ./fs/locks.c.vpid_core 2006-02-02 14:15:34.551876056 +0300
+++ ./fs/locks.c 2006-02-02 14:33:58.801004672 +0300
@@ -1573,7 +1573,7 @@ int fcntl_getlk(struct file *filp, struc

  flock.l_type = F_UNLCK;
  if (fl != NULL) {
-  flock.l_pid = fl->fl_pid;
+  flock.l_pid = pid_type_to_vpid(PIDTYPE_TGID, fl->fl_pid);
 #if BITS_PER_LONG == 32
   /*
    * Make sure we can represent the posix lock via
@@ -1727,7 +1727,7 @@ int fcntl_getlk64(struct file *filp, str

  flock.l_type = F_UNLCK;
  if (fl != NULL) {
-  flock.l_pid = fl->fl_pid;
+  flock.l_pid = pid_type_to_vpid(PIDTYPE_TGID, fl->fl_pid);
   flock.l_start = fl->fl_start;
   flock.l_len = fl->fl_end == OFFSET_MAX ? 0 :
    fl->fl_end - fl->fl_start + 1;
```

```
--- ./fs/proc/array.c.vpid_core 2006-02-02 14:15:34.661859336 +0300
+++ ./fs/proc/array.c 2006-02-02 14:33:58.802004520 +0300
@@ -174,9 +174,10 @@ 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) ? p->group_leader->real_parent->tgid : 0,
-  pid_alive(p) && p->ptrace ? p->parent->pid : 0,
+        get_task_tgid(p),
+  get_task_pid(p),
+  get_task_ppid(p),
+  pid_alive(p) && p->ptrace ? get_task_pid(p->parent) : 0,
   p->uid, p->euid, p->suid, p->fsuid,
   p->gid, p->egid, p->sgid, p->fsgid);
  read_unlock(&tasklist_lock);
@@ -370,11 +371,12 @@ static int do_task_stat(struct task_stru
  }
  if (task->signal) {
   if (task->signal->tty) {
-   tty_pgrp = task->signal->tty->pgrp;
+   tty_pgrp = pid_type_to_vpid(PIDTYPE_PGID,
+     task->signal->tty->pgrp);
    tty_nr = new_encode_dev(tty_devnum(task->signal->tty));
   }
-  pgid = process_group(task);
-  sid = task->signal->session;
+  pgid = get_task_pgid(task);
+  sid = get_task_sid(task);
   cmin_flt = task->signal->cmin_flt;
   cmaj_flt = task->signal->cmaj_flt;
   cutime = task->signal->cutime;
@@ -388,7 +390,7 @@ static int do_task_stat(struct task_stru
   }
   it_real_value = task->signal->real_timer.expires;
  }
- ppid = pid_alive(task) ? task->group_leader->real_parent->tgid : 0;
+ ppid = get_task_ppid(task);
  read_unlock(&tasklist_lock);

  if (!whole || num_threads<2)
@@ -415,7 +417,7 @@ static int do_task_stat(struct task_stru
  res = sprintf(buffer,"%d (%s) %c %d %d %d %d %d %lu %lu \
 %lu %lu %lu %lu %lu %ld %ld %ld %ld %d %ld %llu %lu %ld %lu %lu %lu %lu %lu \
 %lu %lu %lu %lu %lu %lu %lu %lu %d %d %lu %lu\n",
-  task->pid,
+  get_task_pid(task),
   tcomm,
```

```
   state,
   ppid,
--- ./fs/proc/base.c.vpid_core 2006-02-02 14:15:34.662859184 +0300
+++ ./fs/proc/base.c 2006-02-02 14:33:58.804004216 +0300
@@ -1879,14 +1879,14 @@ static int proc_self_readlink(struct den
         int buflen)
 {
  char tmp[30];
- sprintf(tmp, "%d", current->tgid);
+ sprintf(tmp, "%d", get_task_tgid(current));
  return vfs_readlink(dentry,buffer,buflen,tmp);
 }

 static void *proc_self_follow_link(struct dentry *dentry, struct nameidata *nd)
 {
  char tmp[30];
- sprintf(tmp, "%d", current->tgid);
+ sprintf(tmp, "%d", get_task_tgid(current));
  return ERR_PTR(vfs_follow_link(nd,tmp));
 }

@@ -2134,7 +2134,7 @@ static int get_tid_list(int index, unsig
   * via next_thread().
   */
  if (pid_alive(task)) do {
-  int tid = task->pid;
+  int tid = get_task_pid(task);

   if (--index >= 0)
    continue;
--- ./include/net/scm.h.vpid_core 2006-01-03 06:21:10.000000000 +0300
+++ ./include/net/scm.h 2006-02-02 14:33:58.805004064 +0300
@@ -40,7 +40,7 @@ static __inline__ int scm_send(struct so
  memset(scm, 0, sizeof(*scm));
  scm->creds.uid = current->uid;
  scm->creds.gid = current->gid;
- scm->creds.pid = current->tgid;
+ scm->creds.pid = virt_tgid(current);
  if (msg->msg_controllen <= 0)
   return 0;
  return __scm_send(sock, msg, scm);
--- ./ipc/msg.c.vpid_core 2006-02-02 14:15:35.148785312 +0300
+++ ./ipc/msg.c 2006-02-02 14:33:58.806003912 +0300
@@ -540,7 +540,7 @@ 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 = virt_pid(msr->r_tsk);
     msq->q_rtime = get_seconds();
     wake_up_process(msr->r_tsk);
     smp_mb();
@@ -622,7 +622,7 @@ asmlinkage long sys_msgsnd (int msqid, s
   }
  }

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

  if(!pipelined_send(msq,msg)) {
@@ -718,7 +718,7 @@ asmlinkage long sys_msgrcv (int msqid, s
    list_del(&msg->m_list);
    msq->q_qnum--;
    msq->q_rtime = get_seconds();
-   msq->q_lrpid = current->tgid;
+   msq->q_lrpid = virt_tgid(current);
    msq->q_cbytes -= msg->m_ts;
    atomic_sub(msg->m_ts,&msg_bytes);
    atomic_dec(&msg_hdrs);
--- ./ipc/sem.c.vpid_core 2006-02-02 14:15:35.149785160 +0300
+++ ./ipc/sem.c 2006-02-02 14:33:58.807003760 +0300
@@ -743,7 +743,7 @@ static int semctl_main(int semid, int se
   for (un = sma->undo; un; un = un->id_next)
    un->semadj[semnum] = 0;
   curr->semval = val;
-  curr->sempid = current->tgid;
+  curr->sempid = virt_tgid(current);
   sma->sem_ctime = get_seconds();
   /* maybe some queued-up processes were waiting for this */
   update_queue(sma);
@@ -1135,7 +1135,7 @@ 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, virt_tgid(current));
  if (error <= 0) {
   if (alter && error == 0)
    update_queue (sma);
@@ -1150,7 +1150,7 @@ retry_undos:
  queue.sops = sops;
  queue.nsops = nsops;
  queue.undo = un;
- queue.pid = current->tgid;
+ queue.pid = virt_tgid(current);
```

```
  queue.id = semid;
  queue.alter = alter;
  if (alter)
@@ -1320,7 +1320,7 @@ found:
       sem->semval = 0;
      if (sem->semval > SEMVMX)
       sem->semval = SEMVMX;
-    sem->sempid = current->tgid;
+    sem->sempid = virt_tgid(current);
     }
    }
    sma->sem_otime = get_seconds();
--- ./ipc/shm.c.vpid_core 2006-02-02 14:15:35.149785160 +0300
+++ ./ipc/shm.c 2006-02-02 14:33:58.808003608 +0300
@@ -93,7 +93,7 @@ static inline void shm_inc (int id) {
  if(!(shp = shm_lock(id)))
   BUG();
  shp->shm_atim = get_seconds();
- shp->shm_lprid = current->tgid;
+ shp->shm_lprid = virt_tgid(current);
  shp->shm_nattch++;
  shm_unlock(shp);
 }
@@ -143,7 +143,7 @@ static void shm_close (struct vm_area_st
  /* remove from the list of attaches of the shm segment */
  if(!(shp = shm_lock(id)))
   BUG();
- shp->shm_lprid = current->tgid;
+ shp->shm_lprid = virt_tgid(current);
  shp->shm_dtim = get_seconds();
  shp->shm_nattch--;
  if(shp->shm_nattch == 0 &&
@@ -239,7 +239,7 @@ static int newseg (key_t key, int shmflg
  if(id == -1)
   goto no_id;

- shp->shm_cprid = current->tgid;
+ shp->shm_cprid = virt_tgid(current);
  shp->shm_lprid = 0;
  shp->shm_atim = shp->shm_dtim = 0;
  shp->shm_ctim = get_seconds();
--- ./kernel/capability.c.vpid_core 2006-02-02 14:15:35.152784704 +0300
+++ ./kernel/capability.c 2006-02-02 14:33:58.808003608 +0300
@@ -67,7 +67,7 @@ asmlinkage long sys_capget(cap_user_head
      spin_lock(&task_capability_lock);
      read_lock(&tasklist_lock);

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

```
+     if (pid && pid != virt_pid(current)) {
        target = find_task_by_pid(pid);
        if (!target) {
            ret = -ESRCH;
@@ -100,6 +100,10 @@ static inline int cap_set_pg(int pgrp, k
  int ret = -EPERM;
  int found = 0;

+ pgrp = vpid_to_pid(pgrp);
+ if (pgrp < 0)
+  return ret;
+
  do_each_task_pid(pgrp, PIDTYPE_PGID, g) {
   target = g;
   while_each_thread(g, target) {
@@ -199,7 +203,7 @@ asmlinkage long sys_capset(cap_user_head
      spin_lock(&task_capability_lock);
      read_lock(&tasklist_lock);

-     if (pid > 0 && pid != current->pid) {
+     if (pid > 0 && pid != virt_pid(current)) {
          target = find_task_by_pid(pid);
          if (!target) {
              ret = -ESRCH;
--- ./kernel/exit.c.vpid_core 2006-02-02 14:15:35.156784096 +0300
+++ ./kernel/exit.c 2006-02-02 14:33:58.810003304 +0300
@@ -139,6 +139,8 @@ int session_of_pgrp(int pgrp)
  struct task_struct *p;
  int sid = -1;

+ WARN_ON(is_virtual_pid(pgrp));
+
  read_lock(&tasklist_lock);
  do_each_task_pid(pgrp, PIDTYPE_PGID, p) {
   if (p->signal->session > 0) {
@@ -168,10 +170,12 @@ static int will_become_orphaned_pgrp(int
  struct task_struct *p;
  int ret = 1;

+ WARN_ON(is_virtual_pid(pgrp));
+
  do_each_task_pid(pgrp, PIDTYPE_PGID, p) {
   if (p == ignored_task
     || p->exit_state
-    || p->real_parent->pid == 1)
+    || virt_pid(p->real_parent) == 1)
     continue;
   if (process_group(p->real_parent) != pgrp
```

```
        && p->real_parent->signal->session == p->signal->session) {
@@ -186,6 +190,8 @@ int is_orphaned_pgrp(int pgrp)
 {
  int retval;

+ WARN_ON(is_virtual_pid(pgrp));
+
  read_lock(&tasklist_lock);
  retval = will_become_orphaned_pgrp(pgrp, NULL);
  read_unlock(&tasklist_lock);
@@ -198,6 +204,8 @@ static int has_stopped_jobs(int pgrp)
  int retval = 0;
  struct task_struct *p;

+ WARN_ON(is_virtual_pid(pgrp));
+
  do_each_task_pid(pgrp, PIDTYPE_PGID, p) {
   if (p->state != TASK_STOPPED)
    continue;
@@ -263,6 +271,9 @@ void __set_special_pids(pid_t session, p
 {
  struct task_struct *curr = current->group_leader;

+ WARN_ON(is_virtual_pid(pgrp));
+ WARN_ON(is_virtual_pid(session));
+
  if (curr->signal->session != session) {
   detach_pid(curr, PIDTYPE_SID);
   curr->signal->session = session;
@@ -957,14 +968,19 @@ asmlinkage void sys_exit_group(int error
 static int eligible_child(pid_t pid, int options, task_t *p)
 {
  if (pid > 0) {
-  if (p->pid != pid)
+  if ((is_virtual_pid(pid) ? virt_pid(p) : p->pid) != pid)
    return 0;
  } else if (!pid) {
   if (process_group(p) != process_group(current))
    return 0;
  } else if (pid != -1) {
-  if (process_group(p) != -pid)
-   return 0;
+  if (__is_virtual_pid(-pid)) {
+   if (virt_pgid(p) != -pid)
+    return 0;
+  } else {
+   if (process_group(p) != -pid)
+    return 0;
```

```
+  }
   }

   /*
@@ -1154,7 +1170,7 @@ static int wait_task_zombie(task_t *p, i
    p->exit_state = EXIT_ZOMBIE;
    return retval;
   }
- retval = p->pid;
+ retval = get_task_pid(p);
   if (p->real_parent != p->parent) {
    write_lock_irq(&tasklist_lock);
    /* Double-check with lock held.  */
@@ -1289,7 +1305,7 @@ bail_ref:
   if (!retval && infop)
    retval = put_user(p->uid, &infop->si_uid);
   if (!retval)
-  retval = p->pid;
+  retval = get_task_pid(p);
   put_task_struct(p);

   BUG_ON(!retval);
--- ./kernel/fork.c.vpid_core 2006-02-02 14:15:55.472695608 +0300
+++ ./kernel/fork.c 2006-02-02 14:41:40.806769120 +0300
@@ -854,7 +854,7 @@ asmlinkage long sys_set_tid_address(int
 {
  current->clear_child_tid = tidptr;

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

 /*
--- ./kernel/sched.c.vpid_core 2006-02-02 14:15:55.479694544 +0300
+++ ./kernel/sched.c 2006-02-02 14:33:58.815002544 +0300
@@ -1674,7 +1674,7 @@ asmlinkage void schedule_tail(task_t *pr
  preempt_enable();
 #endif
  if (current->set_child_tid)
-  put_user(current->pid, current->set_child_tid);
+  put_user(virt_pid(current), current->set_child_tid);
 }

 /*
--- ./kernel/signal.c.vpid_core 2006-02-02 14:15:55.481694240 +0300
+++ ./kernel/signal.c 2006-02-02 14:33:58.817002240 +0300
@@ -838,7 +838,7 @@ 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 = virt_pid(current);
    q->info.si_uid = current->uid;
    break;
   case (unsigned long) SEND_SIG_PRIV:
@@ -1159,6 +1159,11 @@ int __kill_pg_info(int sig, struct sigin
  if (pgrp <= 0)
   return -EINVAL;

+ /* Use __vpid_to_pid(). This function is used under write_lock
+  * tasklist_lock. */
+ if (is_virtual_pid(pgrp))
+  pgrp = __vpid_to_pid(pgrp);
+
  success = 0;
  retval = -ESRCH;
  do_each_task_pid(pgrp, PIDTYPE_PGID, p) {
@@ -1254,7 +1259,7 @@ static int kill_something_info(int sig,

   read_lock(&tasklist_lock);
   for_each_process(p) {
-   if (p->pid > 1 && p->tgid != current->tgid) {
+   if (virt_pid(p) > 1 && p->tgid != current->tgid) {
     int err = group_send_sig_info(sig, info, p);
     ++count;
     if (err != -EPERM)
@@ -1564,7 +1569,7 @@ void do_notify_parent(struct task_struct

  info.si_signo = sig;
  info.si_errno = 0;
- info.si_pid = tsk->pid;
+ info.si_pid = get_task_pid_ve(tsk, tsk->parent);
  info.si_uid = tsk->uid;

  /* FIXME: find out whether or not this is supposed to be c*time. */
@@ -1629,7 +1634,7 @@ static void do_notify_parent_cldstop(str

  info.si_signo = SIGCHLD;
  info.si_errno = 0;
- info.si_pid = tsk->pid;
+ info.si_pid = get_task_pid_ve(tsk, parent);
  info.si_uid = tsk->uid;

  /* FIXME: find out whether or not this is supposed to be c*time. */
@@ -1732,7 +1737,7 @@ 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 = virt_pid(current);
  info.si_uid = current->uid;

  /* Let the debugger run.  */
@@ -1957,7 +1962,7 @@ relock:
     info->si_signo = signr;
     info->si_errno = 0;
     info->si_code = SI_USER;
-    info->si_pid = current->parent->pid;
+    info->si_pid = virt_pid(current->parent);
     info->si_uid = current->parent->uid;
    }

@@ -2340,7 +2345,7 @@ 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 = virt_tgid(current);
  info.si_uid = current->uid;

  return kill_something_info(sig, &info, pid);
@@ -2356,12 +2361,12 @@ 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 = virt_tgid(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 || virt_tgid(p) == tgid)) {
   error = check_kill_permission(sig, &info, p);
   /*
    * The null signal is a permissions and process existence
--- ./kernel/sys.c.vpid_core 2006-02-02 14:15:55.482694088 +0300
+++ ./kernel/sys.c 2006-02-02 14:39:46.597131624 +0300
@@ -281,7 +281,7 @@ asmlinkage long sys_setpriority(int whic
  switch (which) {
   case PRIO_PROCESS:
    if (!who)
-    who = current->pid;
+    who = virt_pid(current);
```

```
    p = find_task_by_pid(who);
    if (p)
     error = set_one_prio(p, niceval, error);
@@ -289,6 +289,8 @@ asmlinkage long sys_setpriority(int whic
   case PRIO_PGRP:
    if (!who)
     who = process_group(current);
+   else
+    who = vpid_to_pid(who);
    do_each_task_pid(who, PIDTYPE_PGID, p) {
     error = set_one_prio(p, niceval, error);
    } while_each_task_pid(who, PIDTYPE_PGID, p);
@@ -334,7 +336,7 @@ asmlinkage long sys_getpriority(int whic
  switch (which) {
   case PRIO_PROCESS:
    if (!who)
-    who = current->pid;
+    who = virt_pid(current);
    p = find_task_by_pid(who);
    if (p) {
     niceval = 20 - task_nice(p);
@@ -345,6 +347,8 @@ asmlinkage long sys_getpriority(int whic
   case PRIO_PGRP:
    if (!who)
     who = process_group(current);
+   else
+    who = vpid_to_pid(who);
    do_each_task_pid(who, PIDTYPE_PGID, p) {
     niceval = 20 - task_nice(p);
     if (niceval > retval)
@@ -1100,9 +1104,10 @@ asmlinkage long sys_setpgid(pid_t pid, p
  struct task_struct *p;
  struct task_struct *group_leader = current->group_leader;
  int err = -EINVAL;
+ pid_t _pgid;

  if (!pid)
-  pid = group_leader->pid;
+  pid = virt_pid(group_leader);
  if (!pgid)
   pgid = pid;
  if (pgid < 0)
@@ -1139,12 +1144,15 @@ asmlinkage long sys_setpgid(pid_t pid, p
  if (p->signal->leader)
   goto out;

+ _pgid = virt_pid(p);
  if (pgid != pid) {
```

```
   struct task_struct *p;

   do_each_task_pid(pgid, PIDTYPE_PGID, p) {
-    if (p->signal->session == group_leader->signal->session)
+    if (p->signal->session == group_leader->signal->session) {
+     _pgid = virt_pid(p);
      goto ok_pgid;
+    }
   } while_each_task_pid(pgid, PIDTYPE_PGID, p);
   goto out;
  }
@@ -1157,7 +1165,14 @@ ok_pgid:
  if (process_group(p) != pgid) {
   detach_pid(p, PIDTYPE_PGID);
   p->signal->pgrp = pgid;
+  set_virt_pgid(p, _pgid);
   attach_pid(p, PIDTYPE_PGID, pgid);
+  if (atomic_read(&p->signal->count) != 1) {
+   task_t *t;
+   for (t = next_thread(p); t != p; t = next_thread(t)) {
+    set_virt_pgid(t, _pgid);
+   }
+  }
  }

  err = 0;
@@ -1170,7 +1185,7 @@ out:
 asmlinkage long sys_getpgid(pid_t pid)
 {
  if (!pid) {
-  return process_group(current);
+  return virt_pgid(current);
  } else {
   int retval;
   struct task_struct *p;
@@ -1182,7 +1197,7 @@ asmlinkage long sys_getpgid(pid_t pid)
   if (p) {
    retval = security_task_getpgid(p);
    if (!retval)
-    retval = process_group(p);
+    retval = virt_pgid(p);
   }
   read_unlock(&tasklist_lock);
   return retval;
@@ -1194,7 +1209,7 @@ 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 virt_pgid(current);
 }

 #endif
@@ -1202,7 +1217,7 @@ asmlinkage long sys_getpgrp(void)
 asmlinkage long sys_getsid(pid_t pid)
 {
  if (!pid) {
-  return current->signal->session;
+  return virt_sid(current);
  } else {
   int retval;
   struct task_struct *p;
@@ -1214,7 +1229,7 @@ asmlinkage long sys_getsid(pid_t pid)
   if(p) {
    retval = security_task_getsid(p);
    if (!retval)
-    retval = p->signal->session;
+    retval = virt_sid(p);
   }
   read_unlock(&tasklist_lock);
   return retval;
@@ -1236,9 +1251,11 @@ asmlinkage long sys_setsid(void)

  group_leader->signal->leader = 1;
  __set_special_pids(group_leader->pid, group_leader->pid);
+ set_virt_pgid(group_leader, virt_pid(current));
+ set_virt_sid(group_leader, virt_pid(current));
  group_leader->signal->tty = NULL;
  group_leader->signal->tty_old_pgrp = 0;
- err = process_group(group_leader);
+ err = virt_pgid(group_leader);
 out:
  write_unlock_irq(&tasklist_lock);
  up(&tty_sem);
--- ./kernel/timer.c.vpid_core 2006-02-02 14:15:35.185779688 +0300
+++ ./kernel/timer.c 2006-02-02 14:33:58.821001632 +0300
@@ -943,7 +943,7 @@ asmlinkage unsigned long sys_alarm(unsig
  */
 asmlinkage long sys_getpid(void)
 {
- return current->tgid;
+ return virt_tgid(current);
 }

 /*
@@ -970,7 +970,7 @@ asmlinkage long sys_getppid(void)
```

```
  parent = me->group_leader->real_parent;
  for (;;) {
-  pid = parent->tgid;
+  pid = virt_tgid(parent);
 #if defined(CONFIG_SMP) || defined(CONFIG_PREEMPT)
 {
   struct task_struct *old = parent;
@@ -1117,7 +1117,7 @@ EXPORT_SYMBOL(schedule_timeout_uninterru
 /* Thread ID - the internal kernel "pid" */
 asmlinkage long sys_gettid(void)
 {
- return current->pid;
+ return virt_pid(current);
 }

 /*
--- ./net/core/scm.c.vpid_core 2006-02-02 14:15:35.252769504 +0300
+++ ./net/core/scm.c 2006-02-02 14:33:58.822001480 +0300
@@ -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 == virt_tgid(current) || capable(CAP_SYS_ADMIN)) &&
      ((creds->uid == current->uid || creds->uid == current->euid ||
        creds->uid == current->suid) || capable(CAP_SETUID)) &&
      ((creds->gid == current->gid || creds->gid == current->egid ||
--- ./net/unix/af_unix.c.vpid_core 2006-02-02 14:15:35.505731048 +0300
+++ ./net/unix/af_unix.c 2006-02-02 14:33:58.823001328 +0300
@@ -439,7 +439,7 @@ 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 = virt_tgid(current);
  sk->sk_peercred.uid = current->euid;
  sk->sk_peercred.gid = current->egid;
  err = 0;
@@ -1043,7 +1043,7 @@ 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 = virt_tgid(current);
  newsk->sk_peercred.uid = current->euid;
  newsk->sk_peercred.gid = current->egid;
  newu = unix_sk(newsk);
```

```
@@ -1107,7 +1107,7 @@ 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 = virt_tgid(current);
  ska->sk_peercred.uid = skb->sk_peercred.uid = current->euid;
  ska->sk_peercred.gid = skb->sk_peercred.gid = current->egid;
```

Subject: [RFC][PATCH 3/7] VPIDs: fork modifications
Posted by dev on Thu, 02 Feb 2006 16:24:09 GMT

View Forum Message <> Reply to Message

This patch changes forking procedure. Basically it adds a function
do_fork_pid() that alows forking task with predefined virtual pid.
Also it adds required vpid manipulations on fork().
Simple and straightforward.

```
--- ./kernel/fork.c.vpid_fork 2006-02-02 14:33:58.811003152 +0300
+++ ./kernel/fork.c 2006-02-02 14:41:40.806769120 +0300
@@ -871,7 +871,7 @@ static task_t *copy_process(unsigned lon
       unsigned long stack_size,
       int __user *parent_tidptr,
       int __user *child_tidptr,
-      int pid)
+      int pid, int vpid)
 {
  int retval;
  struct task_struct *p = NULL;
@@ -932,9 +932,11 @@ static task_t *copy_process(unsigned lon
  p->did_exec = 0;
  copy_flags(clone_flags, p);
  p->pid = pid;
+ if (set_virt_pid(p, alloc_vpid(p->pid, vpid ? : -1)) < 0)
+  goto bad_fork_cleanup_module;
  retval = -EFAULT;
  if (clone_flags & CLONE_PARENT_SETTID)
-  if (put_user(p->pid, parent_tidptr))
+  if (put_user(virt_pid(p), parent_tidptr))
     goto bad_fork_cleanup;

  p->proc_dentry = NULL;
@@ -985,8 +987,13 @@ static task_t *copy_process(unsigned lon
 #endif

  p->tgid = p->pid;
- if (clone_flags & CLONE_THREAD)
```

```
+ set_virt_tgid(p, virt_pid(p));
+ set_virt_pgid(p, virt_pgid(current));
+ set_virt_sid(p, virt_sid(current));
+ if (clone_flags & CLONE_THREAD) {
   p->tgid = current->tgid;
+  set_virt_tgid(p, virt_tgid(current));
+ }

  if ((retval = security_task_alloc(p)))
   goto bad_fork_cleanup_policy;
@@ -1181,6 +1188,9 @@ bad_fork_cleanup_cpuset:
 #endif
  cpuset_exit(p);
 bad_fork_cleanup:
+ if (virt_pid(p) != p->pid && virt_pid(p) > 0)
+  free_vpid(virt_pid(p));
+bad_fork_cleanup_module:
  if (p->binfmt)
   module_put(p->binfmt->module);
 bad_fork_cleanup_put_domain:
@@ -1206,7 +1216,7 @@ task_t * __devinit fork_idle(int cpu)
  task_t *task;
  struct pt_regs regs;

- task = copy_process(CLONE_VM, 0, idle_regs(&regs), 0, NULL, NULL, 0);
+ task = copy_process(CLONE_VM, 0, idle_regs(&regs), 0, NULL, NULL, 0, 0);
  if (!task)
   return ERR_PTR(-ENOMEM);
  init_idle(task, cpu);
@@ -1236,12 +1246,12 @@ static inline int fork_traceflag (unsign
  * It copies the process, and if successful kick-starts
  * it and waits for it to finish using the VM if required.
  */
-long do_fork(unsigned long clone_flags,
+static long do_fork_pid(unsigned long clone_flags,
        unsigned long stack_start,
        struct pt_regs *regs,
        unsigned long stack_size,
        int __user *parent_tidptr,
-       int __user *child_tidptr)
+       int __user *child_tidptr, int vpid)
 {
  struct task_struct *p;
  int trace = 0;
@@ -1255,7 +1265,8 @@ long do_fork(unsigned long clone_flags,
   clone_flags |= CLONE_PTRACE;
  }
```

```
- p = copy_process(clone_flags, stack_start, regs, stack_size, parent_tidptr, child_tidptr, pid);
+ p = copy_process(clone_flags, stack_start, regs, stack_size, parent_tidptr,
+   child_tidptr, pid, vpid);
  /*
   * Do this prior waking up the new thread - the thread pointer
   * might get invalid after that point, if the thread exits quickly.
@@ -1298,6 +1309,17 @@ long do_fork(unsigned long clone_flags,
  return pid;
 }

+long do_fork(unsigned long clone_flags,
+  unsigned long stack_start,
+  struct pt_regs *regs,
+  unsigned long stack_size,
+  int __user *parent_tidptr,
+  int __user *child_tidptr)
+{
+ return do_fork_pid(clone_flags, stack_start, regs, stack_size,
+   parent_tidptr, child_tidptr, 0);
+}
+
 #ifndef ARCH_MIN_MMSTRUCT_ALIGN
 #define ARCH_MIN_MMSTRUCT_ALIGN 0
 #endif
```

Subject: [RFC][PATCH 4/7] VPIDs: vpid macros in non-VPID case
Posted by Kirill Korotaev on Thu, 02 Feb 2006 16:26:28 GMT
View Forum Message <> Reply to Message

This patch specifies an "interface" to virtual pids. That is
some macros/inlines to obtain pids from tasks, convert pids
to vpids and vice versa and set pids.

Kirill

```
--- ./include/linux/pid.h.vpid_macro 2006-01-03 06:21:10.000000000 +0300
+++ ./include/linux/pid.h 2006-02-02 14:32:41.162807472 +0300
@@ -19,6 +19,19 @@ struct pid
  struct list_head pid_list;
 };

+#ifndef CONFIG_VIRTUAL_PIDS
+#define __is_virtual_pid(pid)  0
+#define is_virtual_pid(pid)  0
+#define vpid_to_pid(pid)  (pid)
+#define __vpid_to_pid(pid)  (pid)
+#define pid_type_to_vpid(type, pid) (pid)
```

```
+#define __pid_type_to_vpid(type, pid) (pid)
+#define comb_vpid_to_pid(pid)  (pid)
+#define comb_pid_to_vpid(pid)  (pid)
+#define alloc_vpid(pid, vpid)  (pid)
+#define free_vpid(vpid)   do { } while (0)
+#endif
+
 #define pid_task(elem, type) \
  list_entry(elem, struct task_struct, pids[type].pid_list)

--- ./include/linux/sched.h.vpid_macro 2006-02-02 14:15:55.455698192 +0300
+++ ./include/linux/sched.h 2006-02-02 14:32:41.164807168 +0300
@@ -1257,6 +1257,78 @@ static inline unsigned long *end_of_stac

 #endif

+#ifndef CONFIG_VIRTUAL_PIDS
+static inline pid_t virt_pid(struct task_struct *tsk)
+{
+ return tsk->pid;
+}
+
+static inline pid_t virt_tgid(struct task_struct *tsk)
+{
+ return tsk->tgid;
+}
+
+static inline pid_t virt_pgid(struct task_struct *tsk)
+{
+ return tsk->signal->pgrp;
+}
+
+static inline pid_t virt_sid(struct task_struct *tsk)
+{
+ return tsk->signal->session;
+}
+
+static inline pid_t get_task_pid_ve(struct task_struct *tsk,
+  struct task_struct *ve_tsk)
+{
+ return tsk->pid;
+}
+
+static inline pid_t get_task_pid(struct task_struct *tsk)
+{
+ return tsk->pid;
+}
+
```

```
+static inline pid_t get_task_tgid(struct task_struct *tsk)
+{
+ return tsk->tgid;
+}
+
+static inline pid_t get_task_pgid(struct task_struct *tsk)
+{
+ return tsk->signal->pgrp;
+}
+
+static inline pid_t get_task_sid(struct task_struct *tsk)
+{
+ return tsk->signal->session;
+}
+
+static inline int set_virt_pid(struct task_struct *tsk, pid_t pid)
+{
+ return 0;
+}
+
+static inline void set_virt_tgid(struct task_struct *tsk, pid_t pid)
+{
+}
+
+static inline void set_virt_pgid(struct task_struct *tsk, pid_t pid)
+{
+}
+
+static inline void set_virt_sid(struct task_struct *tsk, pid_t pid)
+{
+}
+
+static inline pid_t get_task_ppid(struct task_struct *p)
+{
+ if (!pid_alive(p))
+  return 0;
+ return (p->pid > 1 ? p->group_leader->real_parent->tgid : 0);
+}
+#endif
+
 /* set thread flags in other task's structures
  * - see asm/thread_info.h for TIF_xxxx flags available
  */
```

Subject: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by Kirill Korotaev on Thu, 02 Feb 2006 16:30:16 GMT

This is the main patch which contains all vpid-to-pid conversions
and auxilliary stuff. Virtual pids are distinguished from real ones
by the VPID_BIT bit set. Conversion from vpid to pid and vice versa
is performed in two ways: fast way, when vpid and it's according pid
differ only in VPID_BIT bit set ("linear" case), and more complex way,
when pid may correspond to any vpid ("sparse" case) - in this case we
use a hash-table based mapping.

Note that this patch implies that we have a public vps_info_t pointer
type to represent VPS (otherwise it is useless) so that it can be used
in any virtualisation solution. Virtualization solution should have the
following "interface":
1. vps_info_t has member int id;
2. vps_info_t has member struct task_struct *init_task;
3. the following macros/functions are defined:
a. inside_vps() - returns true if current task is now
inside VPS;
b. task_inside_vps(task_t *) - returns true if task
belongs to VPS;
c. current_vps() - returns vps_info_t for current VPS;
d. task_vps(task_t *) - returns vps_info_t that task
belongs to;
e. set_sparce_vpid(vps_info_t) - switches VPS into state
when "sparce" conversion is used;
f. sparse_vpid(vps_info_t) - returns true if vps is in
"sparce" state and false if it is in "linear";
g. get_vps_tasks_num(vps_info_t) - returns the number of
tasks that belong to VPS.

Kirill

```
--- ./include/linux/pid.h.vpid_virt 2006-02-02 14:32:41.162807472 +0300
+++ ./include/linux/pid.h 2006-02-02 14:33:17.963212960 +0300
@@ -10,11 +10,17 @@ enum pid_type
  PIDTYPE_MAX
 };

+#define VPID_BIT 10
+#define VPID_DIV (1 << VPID_BIT)
+
 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_VIRTUAL_PIDS
+ int vnr;
```

```
+#endif
  /* list of pids with the same nr, only one of them is in the hash */
  struct list_head pid_list;
 };
@@ -30,6 +36,52 @@ struct pid
 #define comb_pid_to_vpid(pid)  (pid)
 #define alloc_vpid(pid, vpid)  (pid)
 #define free_vpid(vpid)   do { } while (0)
+#else /* CONFIG_VIRTUAL_PIDS */
+#define __is_virtual_pid(pid)  ((pid) & VPID_DIV)
+#define is_virtual_pid(pid)  (__is_virtual_pid(pid) || \
+  (((pid) == 1) && inside_vps()))
+
+extern int vpid_to_pid(int pid);
+extern int __vpid_to_pid(int pid);
+extern pid_t pid_type_to_vpid(int type, pid_t pid);
+extern pid_t __pid_type_to_vpid(int type, pid_t pid);
+
+static inline int comb_vpid_to_pid(int vpid)
+{
+ int pid = vpid;
+
+ if (vpid > 0) {
+  pid = vpid_to_pid(vpid);
+  if (unlikely(pid < 0))
+   return 0;
+ } else if (vpid < 0) {
+  pid = vpid_to_pid(-vpid);
+  if (unlikely(pid < 0))
+   return 0;
+  pid = -pid;
+ }
+ return pid;
+}
+
+static inline int comb_pid_to_vpid(int pid)
+{
+ int vpid = pid;
+
+ if (pid > 0) {
+  vpid = pid_type_to_vpid(PIDTYPE_PID, pid);
+  if (unlikely(vpid < 0))
+   return 0;
+ } else if (pid < 0) {
+  vpid = pid_type_to_vpid(PIDTYPE_PGID, -pid);
+  if (unlikely(vpid < 0))
+   return 0;
+  vpid = -vpid;
```

```
+ }
+ return vpid;
+}
+
+extern int alloc_vpid(int pid, int vpid);
+extern void free_vpid(int vpid);
 #endif

 #define pid_task(elem, type) \
--- ./include/linux/sched.h.vpid_virt 2006-02-02 14:32:41.164807168 +0300
+++ ./include/linux/sched.h 2006-02-02 14:58:53.129832160 +0300
@@ -1327,6 +1327,80 @@ static inline pid_t get_task_ppid(struct
   return 0;
  return (p->pid > 1 ? p->group_leader->real_parent->tgid : 0);
 }
+#else
+static inline pid_t virt_pid(struct task_struct *tsk)
+{
+ return tsk->pids[PIDTYPE_PID].vnr;
+}
+
+static inline pid_t virt_tgid(struct task_struct *tsk)
+{
+ return tsk->pids[PIDTYPE_TGID].vnr;
+}
+
+static inline pid_t virt_pgid(struct task_struct *tsk)
+{
+ return tsk->pids[PIDTYPE_PGID].vnr;
+}
+
+static inline pid_t virt_sid(struct task_struct *tsk)
+{
+ return tsk->pids[PIDTYPE_SID].vnr;
+}
+
+static inline pid_t get_task_pid_ve(struct task_struct *tsk,
+  struct task_struct *ve_tsk)
+{
+ return task_inside_vps(ve_tsk) ? virt_pid(tsk) : tsk->pid;
+}
+
+static inline pid_t get_task_pid(struct task_struct *tsk)
+{
+ return inside_vps() ? virt_pid(tsk) : tsk->pid;
+}
+
+static inline pid_t get_task_tgid(struct task_struct *tsk)
```

```
+{
+ return inside_vps() ? virt_tgid(tsk) : tsk->pid;
+}
+
+static inline pid_t get_task_pgid(struct task_struct *tsk)
+{
+ return inside_vps() ? virt_pgid(tsk) : tsk->signal->pgrp;
+}
+
+static inline pid_t get_task_sid(struct task_struct *tsk)
+{
+ return inside_vps() ? virt_sid(tsk) : tsk->signal->session;
+}
+
+static inline int set_virt_pid(struct task_struct *tsk, pid_t pid)
+{
+ tsk->pids[PIDTYPE_PID].vnr = pid;
+ return pid;
+}
+
+static inline void set_virt_tgid(struct task_struct *tsk, pid_t pid)
+{
+ tsk->pids[PIDTYPE_TGID].vnr = pid;
+}
+
+static inline void set_virt_pgid(struct task_struct *tsk, pid_t pid)
+{
+ tsk->pids[PIDTYPE_PGID].vnr = pid;
+}
+
+static inline void set_virt_sid(struct task_struct *tsk, pid_t pid)
+{
+ tsk->pids[PIDTYPE_SID].vnr = pid;
+}
+
+static inline pid_t get_task_ppid(struct task_struct *p)
+{
+ if (!pid_alive(p))
+  return 0;
+ return (get_task_pid(p) > 1) ? get_task_pid(p->real_parent) : 0;
+}
 #endif

 /* set thread flags in other task's structures
--- ./kernel/pid.c.vpid_virt 2006-02-02 14:15:35.165782728 +0300
+++ ./kernel/pid.c 2006-02-02 14:58:34.632644160 +0300
@@ -27,6 +27,14 @@
 #include <linux/bootmem.h>
```

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

+#ifdef CONFIG_VIRTUAL_PIDS
+static void __free_vpid(int vpid, struct task_struct *ve_tsk);
+#define PIDMAP_NRFREE (BITS_PER_PAGE / 2)
+#else
+#define __free_vpid(vpid, tsk) do { } while (0)
+#define PIDMAP_NRFREE BITS_PER_PAGE
+#endif
+
 #define pid_hashfn(nr) hash_long((unsigned long)nr, pidhash_shift)
 static struct hlist_head *pid_hash[PIDTYPE_MAX];
 static int pidhash_shift;
@@ -58,7 +66,7 @@ typedef struct pidmap {
 } pidmap_t;

 static pidmap_t pidmap_array[PIDMAP_ENTRIES] =
-  { [ 0 ... PIDMAP_ENTRIES-1 ] = { ATOMIC_INIT(BITS_PER_PAGE), NULL } };
+  { [ 0 ... PIDMAP_ENTRIES-1 ] = { ATOMIC_INIT(PIDMAP_NRFREE), NULL } };

 static  __cacheline_aligned_in_smp DEFINE_SPINLOCK(pidmap_lock);

@@ -67,6 +75,7 @@ fastcall void free_pidmap(int pid)
  pidmap_t *map = pidmap_array + pid / BITS_PER_PAGE;
  int offset = pid & BITS_PER_PAGE_MASK;

+ BUG_ON(__is_virtual_pid(pid) || pid == 1);
  clear_bit(offset, map->page);
  atomic_inc(&map->nr_free);
 }
@@ -77,6 +86,8 @@ int alloc_pidmap(void)
  pidmap_t *map;

  pid = last + 1;
+ if (__is_virtual_pid(pid))
+  pid += VPID_DIV;
  if (pid >= pid_max)
   pid = RESERVED_PIDS;
  offset = pid & BITS_PER_PAGE_MASK;
@@ -107,6 +118,8 @@ int alloc_pidmap(void)
     }
     offset = find_next_offset(map, offset);
     pid = mk_pid(map, offset);
+    if (__is_virtual_pid(pid))
+     pid += VPID_DIV;
    /*
     * find_next_offset() found a bit, the pid from it
     * is in-bounds, and if we fell back to the last
```

```
@@ -127,6 +140,8 @@ int alloc_pidmap(void)
     break;
   }
   pid = mk_pid(map, offset);
+  if (__is_virtual_pid(pid))
+   pid += VPID_DIV;
  }
  return -1;
 }
@@ -201,6 +216,7 @@ void fastcall detach_pid(task_t *task, e
   if (tmp != type && find_pid(tmp, nr))
    return;

+ __free_vpid(task->pids[type].vnr, task);
  free_pidmap(nr);
 }

@@ -234,6 +250,9 @@ void switch_exec_pids(task_t *leader, ta

  leader->pid = leader->tgid = thread->pid;
  thread->pid = thread->tgid;
+ set_virt_tgid(leader, virt_pid(thread));
+ set_virt_pid(leader, virt_pid(thread));
+ set_virt_pid(thread, virt_tgid(thread));

  attach_pid(thread, PIDTYPE_PID, thread->pid);
  attach_pid(thread, PIDTYPE_TGID, thread->tgid);
@@ -247,6 +266,344 @@ void switch_exec_pids(task_t *leader, ta
  attach_pid(leader, PIDTYPE_SID, leader->signal->session);
 }

+#ifdef CONFIG_VIRTUAL_PIDS
+/* Virtual PID bits.
+ *
+ * At the moment all internal structures in kernel store real global pid.
+ * The only place, where virtual PID is used, is at user frontend. We
+ * remap virtual pids obtained from user to global ones (vpid_to_pid) and
+ * map globals to virtuals before showing them to user (virt_pid_type).
+ *
+ * We hold virtual PIDs inside struct pid, so map global -> virtual is easy.
+ */
+
+pid_t __pid_type_to_vpid(int type, pid_t pid)
+{
+ struct pid * p;
+
+ if (unlikely(is_virtual_pid(pid)))
+  return -1;
```

```
+
+ read_lock(&tasklist_lock);
+ p = find_pid(type, pid);
+ if (p) {
+  pid = p->vnr;
+ } else {
+  pid = -1;
+ }
+ read_unlock(&tasklist_lock);
+ return pid;
+}
+
+pid_t pid_type_to_vpid(int type, pid_t pid)
+{
+ int vpid;
+
+ if (unlikely(pid <= 0))
+  return pid;
+
+ BUG_ON(is_virtual_pid(pid));
+
+ if (!inside_vps())
+  return pid;
+
+ vpid = __pid_type_to_vpid(type, pid);
+ if (unlikely(vpid == -1)) {
+  /* It is allowed: global pid can be used everywhere.
+   * This can happen, when kernel remembers stray pids:
+   * signal queues, locks etc.
+   */
+  vpid = pid;
+ }
+ return vpid;
+}
+
+/* To map virtual pids to global we maintain special hash table.
+ *
+ * Mapping entries are allocated when a process with non-trivial
+ * mapping is forked, which is possible only after VE migrated.
+ * Mappings are destroyed, when a global pid is removed from global
+ * pidmap, which means we do not need to refcount mappings.
+ */
+
+static struct hlist_head *vpid_hash;
+
+struct vpid_mapping
+{
+ int     pid;
```

```
+ int     vpid;
+ int     vpsid;
+ struct hlist_node link;
+};
+
+static kmem_cache_t *vpid_mapping_cachep;
+
+static inline int vpid_hashfn(int vnr, int vpsid)
+{
+ return hash_long((unsigned long)(vnr + (vpsid << 16)), pidhash_shift);
+}
+
+struct vpid_mapping *__lookup_vpid_mapping(int vnr, int vpsid)
+{
+ struct hlist_node *elem;
+ struct vpid_mapping *map;
+
+ hlist_for_each_entry(map, elem,
+   &vpid_hash[vpid_hashfn(vnr, vpsid)], link) {
+  if (map->vpid == vnr && map->vpsid == vpsid)
+   return map;
+ }
+ return NULL;
+}
+
+/* __vpid_to_pid() is raw version of vpid_to_pid(). It is to be used
+ * only under tasklist_lock. In some places we must use only this version
+ * (f.e. __kill_pg_info is called under write lock!)
+ *
+ * Caller should pass virtual pid. This function returns an error, when
+ * seeing a global pid.
+ */
+int __vpid_to_pid(int pid)
+{
+ struct vpid_mapping *map;
+ vps_info_t vps;
+
+ if (unlikely(!is_virtual_pid(pid) || !inside_vps()))
+  return -1;
+
+ vps = current_vps();
+ if (!sparse_vpid(vps)) {
+  if (pid != 1)
+   return pid - VPID_DIV;
+  return vps->init_task->pid;
+ }
+
+ map = __lookup_vpid_mapping(pid, vps->id);
```

```
+ if (map)
+  return map->pid;
+ return -1;
+}
+
+int vpid_to_pid(int pid)
+{
+ /* User gave bad pid. It is his problem. */
+ if (unlikely(pid <= 0))
+  return pid;
+
+ if (!is_virtual_pid(pid))
+  return pid;
+
+ read_lock(&tasklist_lock);
+ pid = __vpid_to_pid(pid);
+ read_unlock(&tasklist_lock);
+ return pid;
+}
+
+/*
+ * In simple case we have trivial vpid-to-pid conversion rule:
+ * vpid == 1 -> vps->init_task->pid
+ * else         pid & ~VPID_DIV
+ *
+ * when things get more complex we need to allocate mappings...
+ */
+
+static int add_mapping(int pid, int vpid, int vpsid, struct hlist_head *cache)
+{
+ if (pid > 0 && vpid > 0 && !__lookup_vpid_mapping(vpid, vpsid)) {
+  struct vpid_mapping *m;
+
+  if (hlist_empty(cache)) {
+   m = kmem_cache_alloc(vpid_mapping_cachep, GFP_ATOMIC);
+   if (unlikely(m == NULL))
+    return -ENOMEM;
+  } else {
+   m = hlist_entry(cache->first, struct vpid_mapping,
+     link);
+   hlist_del(&m->link);
+  }
+  m->pid = pid;
+  m->vpid = vpid;
+  m->vpsid = vpsid;
+  hlist_add_head(&m->link,
+    &vpid_hash[vpid_hashfn(vpid, vpsid)]);
+ }
```

```
+ return 0;
+}
+
+static int switch_to_sparse_mapping(int pid, vps_info_t vps)
+{
+ struct hlist_head cache;
+ task_t *g, *t;
+ int pcount;
+ int err;
+
+ /* Transition happens under write_lock_irq, so we try to make
+  * it more reliable and fast preallocating mapping entries.
+  * pcounter may be not enough, we could have lots of orphaned
+  * process groups and sessions, which also require mappings.
+  */
+ INIT_HLIST_HEAD(&cache);
+ pcount = get_vps_tasks_num(vps);
+ err = -ENOMEM;
+ while (pcount > 0) {
+  struct vpid_mapping *m;
+  m = kmem_cache_alloc(vpid_mapping_cachep, GFP_KERNEL);
+  if (!m)
+   goto out;
+  hlist_add_head(&m->link, &cache);
+  pcount--;
+ }
+
+ write_lock_irq(&tasklist_lock);
+ err = 0;
+ if (sparse_vpid(vps))
+  goto out_sparse;
+
+ err = -ENOMEM;
+ do_each_thread(g, t) {
+  if (t->pid == pid)
+   continue;
+  if (add_mapping(t->pid, virt_pid(t), vps->id, &cache))
+   goto out_unlock;
+ } while_each_thread(g, t);
+
+ for_each_process(t) {
+  if (t->pid == pid)
+   continue;
+
+  if (add_mapping(t->tgid, virt_tgid(t), vps->id,
+      &cache))
+   goto out_unlock;
+  if (add_mapping(t->signal->pgrp, virt_pgid(t), vps->id,
```

```
+     &cache))
+   goto out_unlock;
+  if (add_mapping(t->signal->session, virt_sid(t), vps->id,
+     &cache))
+   goto out_unlock;
+ }
+ set_sparse_vpid(vps);
+ err = 0;
+
+out_unlock:
+ if (err) {
+  int i;
+
+  for (i=0; i<(1<<pidhash_shift); i++) {
+   struct hlist_node *elem, *next;
+   struct vpid_mapping *map;
+
+   hlist_for_each_entry_safe(map, elem, next, &vpid_hash[i], link) {
+    if (map->vpsid == vps->id) {
+     hlist_del(elem);
+     hlist_add_head(elem, &cache);
+    }
+   }
+  }
+ }
+out_sparse:
+ write_unlock_irq(&tasklist_lock);
+
+out:
+ while (!hlist_empty(&cache)) {
+  struct vpid_mapping *m;
+  m = hlist_entry(cache.first, struct vpid_mapping, link);
+  hlist_del(&m->link);
+  kmem_cache_free(vpid_mapping_cachep, m);
+ }
+ return err;
+}
+
+int alloc_vpid(int pid, int virt_pid)
+{
+ int result;
+ struct vpid_mapping *m;
+ vps_info_t vps;
+
+ if (!inside_vps())
+  return pid;
+
+ vps = current_vps();
```

```
+ if (!sparse_vpid(vps)) {
+  if (virt_pid == -1)
+   return pid + VPID_DIV;
+
+  if (virt_pid == 1 || virt_pid == pid + VPID_DIV)
+   return virt_pid;
+
+  if ((result = switch_to_sparse_mapping(pid, vps)) < 0)
+   return result;
+ }
+
+ m = kmem_cache_alloc(vpid_mapping_cachep, GFP_KERNEL);
+ if (!m)
+  return -ENOMEM;
+
+ m->pid = pid;
+ m->vpsid = vps->id;
+
+ result = (virt_pid == -1) ? pid + VPID_DIV : virt_pid;
+
+ write_lock_irq(&tasklist_lock);
+ if (unlikely(__lookup_vpid_mapping(result, m->vpsid))) {
+  if (virt_pid > 0) {
+   result = -EEXIST;
+   goto out;
+  }
+
+  /* No luck. Now we search for some not-existing vpid.
+   * It is weak place. We do linear search. */
+  do {
+   result++;
+   if (!__is_virtual_pid(result))
+    result += VPID_DIV;
+   if (result >= pid_max)
+    result = RESERVED_PIDS + VPID_DIV;
+  } while (__lookup_vpid_mapping(result, m->vpsid) != NULL);
+
+  /* And set last_pid in hope future alloc_pidmap to avoid
+   * collisions after future alloc_pidmap() */
+  last_pid = result - VPID_DIV;
+ }
+ if (result > 0) {
+  m->vpid = result;
+  hlist_add_head(&m->link,
+    &vpid_hash[vpid_hashfn(result, m->vpsid)]);
+ }
+out:
+ write_unlock_irq(&tasklist_lock);
```

```
+ if (result < 0)
+  kmem_cache_free(vpid_mapping_cachep, m);
+ return result;
+}
+
+static void __free_vpid(int vpid, struct task_struct *ve_tsk)
+{
+ struct vpid_mapping *m;
+ vps_info_t vps;
+
+ if (!__is_virtual_pid(vpid) && (vpid != 1 || !task_inside_vps(ve_tsk)))
+  return;
+
+ vps = task_vps(ve_tsk);
+ if (!sparse_vpid(vps))
+  return;
+
+ m = __lookup_vpid_mapping(vpid, vps->id);
+ BUG_ON(m == NULL);
+ hlist_del(&m->link);
+ kmem_cache_free(vpid_mapping_cachep, m);
+}
+EXPORT_SYMBOL(alloc_vpid);
+
+void free_vpid(int vpid)
+{
+ write_lock_irq(&tasklist_lock);
+ __free_vpid(vpid, current);
+ write_unlock_irq(&tasklist_lock);
+}
+
+#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
@@ -273,6 +630,14 @@ void __init pidhash_init(void)
   for (j = 0; j < pidhash_size; j++)
    INIT_HLIST_HEAD(&pid_hash[i][j]);
  }
+
+#ifdef CONFIG_VIRTUAL_PIDS
+ vpid_hash = alloc_bootmem(pidhash_size * sizeof(struct hlist_head));
+ if (!vpid_hash)
+  panic("Could not alloc vpid_hash!\n");
+ for (j = 0; j < pidhash_size; j++)
+  INIT_HLIST_HEAD(&vpid_hash[j]);
+#endif
```

```
 }

 void __init pidmap_init(void)
@@ -289,4 +654,12 @@ void __init pidmap_init(void)

  for (i = 0; i < PIDTYPE_MAX; i++)
   attach_pid(current, i, 0);
+
+#ifdef CONFIG_VIRTUAL_PIDS
+ vpid_mapping_cachep =
+  kmem_cache_create("vpid_mapping",
+    sizeof(struct vpid_mapping),
+    __alignof__(struct vpid_mapping),
+    SLAB_PANIC, NULL, NULL);
+#endif
 }
```

Subject: [RFC][PATCH 6/7] VPIDs: small proc VPID export
Posted by Kirill Korotaev on Thu, 02 Feb 2006 16:31:48 GMT

View Forum Message <> Reply to Message

Export of task VPID to /proc
Simple.

Kirill

```
--- ./fs/proc/array.c.vpid_proc 2006-02-02 14:33:58.000000000 +0300
+++ ./fs/proc/array.c 2006-02-02 17:54:34.673273968 +0300
@@ -200,6 +200,8 @@ static inline char * task_state(struct t
  put_group_info(group_info);

  buffer += sprintf(buffer, "\n");
+
+ buffer += sprintf(buffer, "VPid:\t%d\n", virt_pid(p));
  return buffer;
 }
```

Subject: [RFC][PATCH 7/7] VPIDs: required VPS interface for VPIDs
Posted by Kirill Korotaev on Thu, 02 Feb 2006 16:32:56 GMT

View Forum Message <> Reply to Message

This patch adds small VPS/containers interface which is used by VPIDs

Kirill

```
--- ./include/linux/sched.h.vps_info 2006-02-02 14:58:53.000000000 +0300
+++ ./include/linux/sched.h 2006-02-02 18:38:15.134903248 +0300
@@ -15,6 +15,7 @@
 #include <linux/cpumask.h>
 #include <linux/errno.h>
 #include <linux/nodemask.h>
+#include <linux/vps_info.h>

 #include <asm/system.h>
 #include <asm/semaphore.h>
--- ./include/linux/vps_info.h.vps_info 2006-02-02 18:12:49.500834880 +0300
+++ ./include/linux/vps_info.h 2006-02-02 18:55:41.137886624 +0300
@@ -0,0 +1,44 @@
+#ifndef __VPS_INFO_H_
+#define __VPS_INFO_H_
+
+#include <linux/config.h>
+
+struct vps_common_info {
+ int id;
+ struct task_struct *init_task;
+ int is_sparse;
+ atomic_t counter;
+ atomic_t pcounter;
+};
+
+typedef struct vps_common_info *vps_info_t;
+
+#ifndef CONFIG_VPS
+extern struct vps_common_info super_vps_info;
+
+#define super_vps()  (&super_vps_info)
+#define current_vps()  (super_vps())
+#define task_vps(t)  (super_vps())
+#define inside_vps()  (0)
+#define task_inside_vps(t) (0)
+#define vps_stop(vps)  do { } while (0)
+#define vps_free(vps)  do { } while (0)
+#endif
+
+#define sparse_vpid(vps) ((vps)->is_sparse)
+#define set_sparse_vpid(vps) do { (vps)->is_sparse = 1; } while (0)
+
+#define vps_get(vps)  atomic_inc(&(vps)->counter)
+#define vps_put(vps)  do {     \
+  if (atomic_dec_and_test(&(vps)->counter)) \
+    vps_free(vps);    \
+ } while (0)
```

```
+
+#define vps_task_add(vps) atomic_inc(&(vps)->pcounter)
+#define vps_task_del(vps) do {     \
+  if (atomic_dec_and_test(&(vps)->pcounter)) \
+   vps_stop(vps);   \
+ } while (0)
+#define get_vps_tasks_num(vps) atomic_read(&(vps)->pcounter)
+
+#endif
--- ./init/main.c.vps_info 2006-02-02 14:15:35.000000000 +0300
+++ ./init/main.c 2006-02-02 18:40:36.592398440 +0300
@@ -47,6 +47,7 @@
 #include <linux/rmap.h>
 #include <linux/mempolicy.h>
 #include <linux/key.h>
+#include <linux/vps_info.h>

 #include <asm/io.h>
 #include <asm/bugs.h>
@@ -434,6 +435,14 @@ void __init parse_early_param(void)
  done = 1;
 }

+struct vps_common_info super_vps_info = {
+ .id  = 0,
+ .init_task = &init_task,
+ .counter = ATOMIC_INIT(1),
+ .pcounter = ATOMIC_INIT(1), /* init_task */
+ .is_sparse = 0,
+};
+
 /*
  * Activate the first processor.
  */
--- ./kernel/exit.c.vps_info 2006-02-02 14:33:58.000000000 +0300
+++ ./kernel/exit.c 2006-02-02 18:33:10.953145904 +0300
@@ -31,6 +31,7 @@
 #include <linux/signal.h>
 #include <linux/cn_proc.h>
 #include <linux/mutex.h>
+#include <linux/vps_info.h>

 #include <asm/uaccess.h>
 #include <asm/unistd.h>
@@ -107,6 +108,7 @@ repeat:
  spin_unlock(&p->proc_lock);
  proc_pid_flush(proc_dentry);
  release_thread(p);
```

```
+ vps_task_del(task_vps(p));
  put_task_struct(p);

  p = leader;
--- ./kernel/fork.c.vps_info 2006-02-02 14:41:40.000000000 +0300
+++ ./kernel/fork.c 2006-02-02 18:32:07.910729808 +0300
@@ -44,6 +44,7 @@
 #include <linux/rmap.h>
 #include <linux/acct.h>
 #include <linux/cn_proc.h>
+#include <linux/vps_info.h>

 #include <asm/pgtable.h>
 #include <asm/pgalloc.h>
@@ -1139,6 +1140,7 @@ static task_t *copy_process(unsigned lon
  p->ioprio = current->ioprio;

  SET_LINKS(p);
+ vps_task_add(task_vps(p));
  if (unlikely(p->ptrace & PT_PTRACED))
   __ptrace_link(p, current->parent);

--- ./kernel/pid.c.vps_info 2006-02-02 14:58:34.000000000 +0300
+++ ./kernel/pid.c 2006-02-02 18:54:25.506384360 +0300
@@ -26,6 +26,7 @@
 #include <linux/init.h>
 #include <linux/bootmem.h>
 #include <linux/hash.h>
+#include <linux/vps_info.h>

 #ifdef CONFIG_VIRTUAL_PIDS
 static void __free_vpid(int vpid, struct task_struct *ve_tsk);
```

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by Dave Hansen on Thu, 02 Feb 2006 17:05:51 GMT

View Forum Message <> Reply to Message

On Thu, 2006-02-02 at 19:30 +0300, Kirill Korotaev wrote:
> This is the main patch which contains all vpid-to-pid conversions
> and auxilliary stuff. Virtual pids are distinguished from real ones
> by the VPID_BIT bit set. Conversion from vpid to pid and vice versa
> is performed in two ways: fast way, when vpid and it's according pid
> differ only in VPID_BIT bit set ("linear" case), and more complex way,
> when pid may correspond to any vpid ("sparse" case) - in this case we
> use a hash-table based mapping.

This is an interesting approach. Could you elaborate a bit on on why

you need the two different approaches? What conditions cause the switch
to the sparse approach?

Also, if you could separate those two approaches out into two different
patches, it would be much easier to get a grasp about what's going on.
One of them is just an optimization, right?

Did you happen to catch Linus's mail about his preferred approach?

 http://marc.theaimsgroup.com/?l=linux-kernel&m=113874154 731279&w=2

-- Dave

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by serue on Thu, 02 Feb 2006 19:29:01 GMT
View Forum Message <> Reply to Message

Quoting Dave Hansen (haveblue@us.ibm.com):
> On Thu, 2006-02-02 at 19:30 +0300, Kirill Korotaev wrote:
> > This is the main patch which contains all vpid-to-pid conversions
> > and auxilliary stuff. Virtual pids are distinguished from real ones
> > by the VPID_BIT bit set. Conversion from vpid to pid and vice versa
> > is performed in two ways: fast way, when vpid and it's according pid
> > differ only in VPID_BIT bit set ("linear" case), and more complex way,
> > when pid may correspond to any vpid ("sparse" case) - in this case we
> > use a hash-table based mapping.
>
> This is an interesting approach. Could you elaborate a bit on on why
> you need the two different approaches? What conditions cause the switch
> to the sparse approach?
>
> Also, if you could separate those two approaches out into two different
> patches, it would be much easier to get a grasp about what's going on.
> One of them is just an optimization, right?
>
> Did you happen to catch Linus's mail about his preferred approach?
>
> http://marc.theaimsgroup.com/?l=linux-kernel&m=113874154 731279&w=2

And in particular, is there any functionality which you could not
implement by using this approach, which you can implement with the
pid_to_vpid approach?

thanks,
-serge

Subject: Re: [RFC][PATCH 3/7] VPIDs: fork modifications
Posted by Cedric Le Goater on Thu, 02 Feb 2006 20:08:13 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:

> This patch changes forking procedure. Basically it adds a function
> do_fork_pid() that alows forking task with predefined virtual pid.
> Also it adds required vpid manipulations on fork().
> Simple and straightforward.

I'd like to see what goes behind that patch, specially if it is used to
restart a process tree. Is it in openvz yet ?

c.

Subject: Re: [RFC][PATCH] VPIDs: Virtualization of PIDs (OpenVZ approach)
Posted by Herbert Poetzl on Fri, 03 Feb 2006 03:01:43 GMT
View Forum Message <> Reply to Message

On Thu, Feb 02, 2006 at 06:54:21PM +0300, Kirill Korotaev wrote:
> [RFC][PATCH] VPIDs: Virtualization of PIDs (OpenVZ approach)
>
> OpenVZ project would like to present another approach for VPIDs
> developed by Alexey Kuznetsov. These patches were heavily tested and
> used for half a year in OpenVZ already.
>
> OpenVZ VPIDs serves the same purpose of VPS (container) checkpointing
> and restore. OpenVZ VPS checkpointing code will be released soon.
>
> These VPIDs patches look less intrusive than IBM's, so I would ask to
> consider it for inclusion. Other projects can benefit from this code
> as well.

well, IMHO this approach lacks a few things which would
be very useful in a mainline pid virtualization, which
pretty much explains why it is relatively small

- hierarchical structure (it's flat, only one level)
- a proper administration scheme
- a 'view' into the child pid spaces
- handling of inter context signalling

and, more important, it does not deal with the existing
issues and error cases, where references to pids, tasks,
task groups and sessions aren't handled properly ...

I think that in real world virtualization scenarios
with hundreds of namespaces those 'imprecisions' will
occasionally lead to very strange and random behaviour
which in many cases will go completely unnoticed.

so I really prefer to cleanup the existing pid handling
first, to avoid big surprises later ...

best,
Herbert

> Patch set consists of 7 patches. The main patches are: -
> diff-vpids-macro, which introduces pid access interface -
> diff-vpids-core, which introduces pid translations where required
> using new interfaces - diff-vpids-virt, which introduces virtual pids
> handling in pid.c and pid mappings for VPSs
>
> Patches are against latest 2.6.16-rc1-git6
>
> Kirill Korotaev,
> OpenVZ team
>
> -
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/

Subject: Re: [RFC][PATCH] VPIDs: Virtualization of PIDs (OpenVZ approach)
Posted by Kirill Korotaev on Fri, 03 Feb 2006 10:30:05 GMT
View Forum Message <> Reply to Message

> well, IMHO this approach lacks a few things which would
> be very useful in a mainline pid virtualization, which
> pretty much explains why it is relatively small
>
> - hierarchical structure (it's flat, only one level)
PID virtualization has nothing to do with containers and its structure.
This VPID patch can be used both in environments with hierarchical and
flat structure.
I mean that this VPID patch introduces some kind of abstraction, which
means that global unique pid is not always what user sees. Thats it.
And kernel should not be modified in all places where access checks
current->pid == allowed_pid are done. Global unique PIDs are preserved.

> - a proper administration scheme
Can't catch what you mean. What kind of administration do you want for

VPIDs? Do you have one for usual PIDs? It is assigned by kernel and
that's it.

> - a 'view' into the child pid spaces
it has. see proc patch.

> - handling of inter context signalling
How is it related to inter context signalling???
virtualization of signaling is a separate task and has nothing to do
with pids.

> and, more important, it does not deal with the existing
> issues and error cases, where references to pids, tasks,
> task groups and sessions aren't handled properly ...
1. if kernel has some errors, these errors should be fixed.
Virtualization doesn't deal with it, doesn't solve such issues, doesn't
make it worse. So what do you mean?
2. if kernel has some issues and error cases can you point them to me? I
will be glad to fix it. Without pointing to the facts your words sound
like a pure speculation. What issues? What error cases? Where task
groups and sessiona aren't handled properly?

> I think that in real world virtualization scenarios
> with hundreds of namespaces those 'imprecisions' will
> occasionally lead to very strange and random behaviour
> which in many cases will go completely unnoticed.
OpenVZ successfully works with >1000 VPSs on a single server.
What scenarios do you mean?
I see no arguments from your side except for some guesses.

> so I really prefer to cleanup the existing pid handling
> first, to avoid big surprises later ...
I'm not against pid handling cleanups, am I?
This can be done in parallel/before/after.

Kirill

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by Alexey Kuznetsov on Fri, 03 Feb 2006 10:52:02 GMT
View Forum Message <> Reply to Message

Hello!

> This is an interesting approach. Could you elaborate a bit on on why
> you need the two different approaches? What conditions cause the switch
> to the sparse approach?
>

> Also, if you could separate those two approaches out into two different
> patches, it would be much easier to get a grasp about what's going on.
> One of them is just an optimization, right?

Exactly. They are not two different "approaches", it is just a simple
optimization.

In our approach each process has pair of pids: one is global unique identifier
(read, it is real pid from viewpoint of kernel), another is virtual pid.
We just do not want to lose cycles allocating both of them all the time,
so we derive virtual pid from global one, it is automatically "virtually"
unique.

Switch to "sparse" mapping happens _only_ after migration, when a process
must be recreated with the same virtual pid, but with new global pid.

> Did you happen to catch Linus's mail about his preferred approach?
>
> http://marc.theaimsgroup.com/?l=linux-kernel&m=113874154 731279&w=2

Of course. Logically, it would be final solution.

VPID approach is pragmatic: it does not modify existing logic, rather
it relies on it. So, it just allows to use virtual pids in a simple
and efficient way, which is enough for all known tasks.

Alexey

Subject: Re: [RFC][PATCH] VPIDs: Virtualization of PIDs (OpenVZ approach)
Posted by Alexey Kuznetsov on Fri, 03 Feb 2006 12:45:58 GMT
View Forum Message <> Reply to Message

Hello!

> - hierarchical structure (it's flat, only one level)

Er... if I understand the question correctly, it is not flat,
it is rather universal.

The main idea of the VPID approach is that it _preserves_ existing
logic completely, all the references, interprocess linkage and process
grouping are made using usual unique pids.

Virtual pids are just the numbers, which user level applications see.
Of course, they depend on the point where you view from.

> - a proper administration scheme

There is nothing to administrate. pids are generated and assigned
automatically.

Virtual pids come to the game only when you move a process from
one host to another, new process is created with do_fork_pid(),
which assignes the same virtual pid and generates brand new global pid,
unique for destination host.

> - a 'view' into the child pid spaces

In order to access a virtual pid space the process must enter
the corresponding container. You are not going to add the whole set
of syscalls, which use not only pid but also a container identifier, right?

For purpose of debugging, vpids are shown /proc/*/status, it is enough.

> - handling of inter context signalling

Not an issue. All the inter-container communication is made using
global pids.

It is necessary to emphasize again, VPID patch _preserves_ currently
existing capabilities: all the processes in all the containers
can be accessed by global pids. F.e. plain "ps axf" executed in "initial
container" shows the whole process tree.

It is one of main points: "virtual" pids are not used to _limit_ access,
is not a "no go" thing, it allows to go on when we have to reinstate
VPS at another host. Access checks are not based on some pids, they
are defined by policy rules, which are checked after corresponding
objects (tasks in our case) are found.

> and, more important, it does not deal with the existing
> issues and error cases, where references to pids, tasks,
> task groups and sessions aren't handled properly ...

I am sorry, I am not aware about such bugs (expect for a few well-known
cases, when someone holds a pid, which can remain stray and be recycled.
But it is apparently orthogonal to the problem under duscussion).
If they do exist they apparently must be fixed asap not depending
on anything else.

Alexey

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by Cedric Le Goater on Fri, 03 Feb 2006 12:48:05 GMT
View Forum Message <> Reply to Message

Alexey Kuznetsov wrote:

>>Did you happen to catch Linus's mail about his preferred approach?
>>
>> http://marc.theaimsgroup.com/?l=linux-kernel&m=113874154 731279&w=2
>
> Of course. Logically, it would be final solution.
>
> VPID approach is pragmatic: it does not modify existing logic, rather
> it relies on it. So, it just allows to use virtual pids in a simple
> and efficient way, which is enough for all known tasks.

And how bad would it be for openvz to move away from the vpid approach ? do you have any plan for it ?

I've also seen that openvz introduces a 'vps_info_t' object, which looks like a some virtualization backend. I'm not sure to have well understood this framework. What the idea behind it ? is it to handle different implementation of the virtualization ?

thanks,

C.

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by Alexey Kuznetsov on Fri, 03 Feb 2006 14:02:29 GMT
View Forum Message <> Reply to Message

Hello!

> And how bad would it be for openvz to move away from the vpid approach ? do
> you have any plan for it ?

Frankly speaking, using pair (container, pid) was the first thing, which we did (year ago), so that from viewpoint of core the switch is not a big deal. :-) However, it was rejected by several reasons:

1. Replacing all the references to pid with pair (container, pid) is quite expensive. F.e. it is possible that a task has a pid from one container,

but it is in process group and/or session of another container,
and its controlling terminal owner by another container. Grr..
So, the structures are bloated, the functions get additional arguments.
And all this is for no real purpose, the functionality comparing with
VPID is even reduced.

2. It is very inconvenient not to see processes inside VPS from host system.
To do ps, strace, gdb etc. we have to move inside VPS. With VPID approach I can
gdb even "init" process of VPS in a way invisible to VPS, see?
Well, and main problem is that gui administration and monotoring tools,
which were existing for ages stop to work and require a major rewrite.
Does it answer to question about plans for moving away?

To summarize: (container, pid) approach looks clean and consistent.
At first sight I loved it, even thought it will solve some of problems
with inter-container access control. But the devil is in details,
I have to learn this again and again: access control must be separate
of real engine, otherwise you get something which does not satisfy anyone.

> I've also seen that openvz introduces a 'vps_info_t' object, which looks
> like a some virtualization backend. I'm not sure to have well understood
> this framework. What the idea behind it ? is it to handle different
> implementation of the virtualization ?

openvz _is_ a complete implementation of virtualization (http://openvz.org),
dealing with... essentially, everything, including even things like
iptables. The VPID patch is just a small part of the whole openvz,
and vps_info_t is just a part of large structure containing data essential
for this problem.

Alexey

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by dev on Fri, 03 Feb 2006 14:03:30 GMT
View Forum Message <> Reply to Message

> I've also seen that openvz introduces a 'vps_info_t' object, which looks
> like a some virtualization backend. I'm not sure to have well understood
> this framework. What the idea behind it ? is it to handle different
> implementation of the virtualization ?
Yes, it was a small container backend, where small piece of
per-container info required for VPIDs is stored.
This patch will be resent in a bit another form, non-related to VPIDs
itself. Something like an abstract container declaration.

Kirill

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by Cedric Le Goater on Fri, 03 Feb 2006 15:40:51 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
>> I've also seen that openvz introduces a 'vps_info_t' object, which looks
>> like a some virtualization backend. I'm not sure to have well understood
>> this framework. What the idea behind it ? is it to handle different
>> implementation of the virtualization ?
>
> Yes, it was a small container backend, where small piece of
> per-container info required for VPIDs is stored.
> This patch will be resent in a bit another form, non-related to VPIDs
> itself. Something like an abstract container declaration.

That is an interesting concept because it could probably be used to satisfy all parties. It seems that everyone has his own idea on what a container should "feel" like, in term of pid virtualisation and container parent hood also, at least. May be there is not a unique model.

Just booted 2.6.15-openvz.025stab014, I'm going to have a look at the concepts you put place.

C.

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by Dave Hansen on Fri, 03 Feb 2006 16:25:18 GMT
View Forum Message <> Reply to Message

On Fri, 2006-02-03 at 17:02 +0300, Alexey Kuznetsov wrote:
> 1. Replacing all the references to pid with pair (container, pid) is quite
> expensive. F.e. it is possible that a task has a pid from one container,
> but it is in process group and/or session of another container,
> and its controlling terminal owner by another container. Grr..
> So, the structures are bloated, the functions get additional arguments.
> And all this is for no real purpose, the functionality comparing with
> VPID is even reduced.

I don't think anything gets bloated, at least in the default case.
Every task points to one and only one container. Things that currently take tasks as arguments still take tasks. If something handling tasks suddenly needs to worry about containers as well, we simply infer the container from the task.

In the very, very rare cases where we can't do that (like a fork() boundary), we _do_ change the APIs to take both task and container. However, these are few and far between, so it isn't that much of a

headache.

In the cases where pids were referenced, and are no longer unique, we
use Eric's weak task references. Right?

-- Dave

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by Kirill Korotaev on Fri, 03 Feb 2006 16:26:45 GMT
View Forum Message <> Reply to Message

> That is an interesting concept because it could probably be used to satisfy
> all parties. It seems that everyone has his own idea on what a container
> should "feel" like, in term of pid virtualisation and container parent hood
> also, at least. May be there is not a unique model.
Yes, it is some concept of abstract container. I will send a series of
patches for this today CCing you, so that you could take a look and
better understand what and how it cab be used for in our opinion.

> Just booted 2.6.15-openvz.025stab014, I'm going to have a look at the
> concepts you put place.
OpenVZ conception is that a VPS is something like a container, with
virtualized TCP/IP, netfilters, rounting, IPC, process tree and so on.
It looks almost exactly like a standalone system. This container is easy
to checkpoint and restore since it has obvious boundaries and is isolated.

Feel free to ask any questions on the forum or directly from me.

Kirill

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by Cedric Le Goater on Fri, 03 Feb 2006 17:05:04 GMT
View Forum Message <> Reply to Message

Alexey Kuznetsov wrote:

> Frankly speaking, using pair (container, pid) was the first thing, which
> we did (year ago), so that from viewpoint of core the switch
> is not a big deal. :-) However, it was rejected by several reasons:
>
> 1. Replacing all the references to pid with pair (container, pid) is quite
> expensive. F.e. it is possible that a task has a pid from one container,
> but it is in process group and/or session of another container,
> and its controlling terminal owner by another container. Grr..

If that happens, it also means your container is not fully isolated which
is also a challenge for the vpid approach when you try to migrate. nop ?

If i take your example with the external process group, what would happen
if the process group leader dies and then you try to migrate that container
? How would you restore the processes in your container that are refering a
dead external process group leader ?

Everything is possible but "loose" isolation on pid raises a lot of issues
on vpids at restart. I would stick to a real strict isolation and forbid
such cases. And, in that case, it's much easier to use the pair approach
(container, pid).

We've been living with the vpid approach also for years and we found issues
that we haven't solve at restart. So we think we might do a better job with
another. But, this still needs to be confirmed :)

> So, the structures are bloated, the functions get additional arguments.
> And all this is for no real purpose, the functionality comparing with
> VPID is even reduced.

i don't see much changes, when you query a task by pid, you only look in
your *current* container pidspace.

some areas in the kernel use directily pids, true. Eric Biederman really
knows well his job on this topic. Many thanks. But, that could be fixed.

> 2. It is very inconvenient not to see processes inside VPS from host system.
> To do ps, strace, gdb etc. we have to move inside VPS. With VPID approach I can
> gdb even "init" process of VPS in a way invisible to VPS, see?

that's another container model issue again. your init process of a VPS
could be the real init. why do you need a fake one ? just trying to
understand all the issues you had to solve and I'm sure they are valid.

> Well, and main problem is that gui administration and monotoring tools,
> which were existing for ages stop to work and require a major rewrite.
> Does it answer to question about plans for moving away?
>
> To summarize: (container, pid) approach looks clean and consistent.
> At first sight I loved it, even thought it will solve some of problems
> with inter-container access control. But the devil is in details,
> I have to learn this again and again: access control must be separate
> of real engine, otherwise you get something which does not satisfy anyone.

hmm, I'm not completely satisfied :) but we'll work this out, we'll find a
way to agree on something.

C.

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by Alexey Kuznetsov on Mon, 06 Feb 2006 09:48:43 GMT
View Forum Message <> Reply to Message

Hello!

> > 1. Replacing all the references to pid with pair (container, pid) is quite
> > expensive. F.e. it is possible that a task has a pid from one container,
> > but it is in process group and/or session of another container,
> > and its controlling terminal owner by another container. Grr..
>
> If that happens, it also means your container is not fully isolated which
> is also a challenge for the vpid approach when you try to migrate. nop ?

Yes, in this case container is not isolated and its migration
is prohibited.

In openvz containers may be not isolated. openvz does not impose
non-overridable access restrictions: f.e. if administrator wants,
he can give access to some raw networking device or to mount some raw
partition inside container fs namespace. Also "vzctl exec" can pass to child
an open file from host (actually, it always passes open pipes to communicate
to the child). Obviously, in all those cases the container cannot migrate.

> If i take your example with the external process group, what would happen
> if the process group leader dies and then you try to migrate that container
> ? How would you restore the processes in your container that are refering a
> dead external process group leader ?

Container can be migrated only when all the references resolve inside
this container. External references are not a normal situation, but
they are not prohibited.

Typical example is:

vzctl exec 202 "sleep 3600 < /dev/null >& /dev/null &"

sleep will be normal process inside container, but its process group
and session are inherited from host system.

```
root@mops:~ # ps axj | fgrep 3600
 4890  6880  6879  6879 ?          -1 S        0   0:00 sleep 3600
root@mops:~ # vzctl exec 202 "ps axj" | fgrep 3600
    1  7904  6879  6879 ?          -1 S        0   0:00 sleep 3600
```

See? pid and ppid look different (virtual ones inside container), but
pgid and sid are not. Apparently, such container cannot migrate
until the sleep exits.

We could force each process visiting container to daemonize and to setsid().
But do not forget that pid space is just a little part of the whole engine,
to force full isolation we have to close all the files opened
in root container, to get rid of its memory space before entering container
etc. But it makes not so much of sense, because in any case we have to keep
at least some way to communicate to host. F.e. even when we allow to pass
only an open pipe, we immediately encounter the situation when a file
owned by one container could refer to processes of another container.

So that, the only way to enforce full isolation is to prohibit
"vzctl exec/enter" as whole.

> We've been living with the vpid approach also for years and we found issues
> that we haven't solve at restart. So we think we might do a better job with
> another. But, this still needs to be confirmed :)

What are the issues?

The only inconvenience which I encountered until now
is a little problem with stray pids. F.e. this happens with flock().
Corresponding kernel structure contains some useless (actually, illegal
and contradicting to the nature of flock()) reference to pid.
If the process took the lock and exited, stray pid remains forever and points
to nowhere. In this case it is silly to prohibit checkpointing,
but we have to restore the flock to a lock with pointing to the same point
in the sky, i.e. to nowhere. With (container, pid) approach we would
restore it pointing to exactly the same empty place in the sky, with
vpids we have to choose a new place. Ugly, but not a real issue.

> i don't see much changes, when you query a task by pid, you only look in
> your *current* container pidspace.

You can query not only task, you can query process group, session,
which is openvz can be not inside the container. You can lookup pid of
file owner, tty session, lock owner etc. Essentially, each place in kernel,
where a pid is stored has to refer to container as well.
Unless, of course, you do not prohibit containers to share
anything and allow them to communicate only via TCP.

> > 2. It is very inconvenient not to see processes inside VPS from host system.

> > To do ps, strace, gdb etc. we have to move inside VPS. With VPID approach I can
> > gdb even "init" process of VPS in a way invisible to VPS, see?
>
> that's another container model issue again. your init process of a VPS
> could be the real init. why do you need a fake one ? just trying to
> understand all the issues you had to solve and I'm sure they are valid.

It is not a fake init, it is a real init. Main goal of openvz is to allow
to start even f.e. the whole instance of stock redhat inside container
including standard init. It is not a strict architectural requirement,
but this option must be present.

BTW the question sounds strange. I would think that in (container, pid)
approach among another limitations you get requirement that you _must_
have a child reaper inside container. With VPIDs this does not matter,
wait() made by parent inside host always returns global pid of child.
But with (container, pid) children can be reaped only inside container, right?

Alexey

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by Alexey Kuznetsov on Mon, 06 Feb 2006 11:24:35 GMT
View Forum Message <> Reply to Message

Hello!

> In the very, very rare cases where we can't do that (like a fork()
> boundary), we _do_ change the APIs to take both task and container.

I promise you much more of boundary cases, unless
you make some windowsish sys_fork_exec_deal_with_all_the_rest(100 arguments)

Look how this works in openvz. It uses pure traditional unixish api.
Fork is not a boundary at all. To enter to a container you
do all the work in steps:

1. change accounting space (sys_setluid())
2. plain fork()
2. tune communication (pipes, ptys etc)
3. chroot()
...
N. enter container (at the moment it is ioctl on a special device, could be
   syscall).

You can omit any step, if you need. You can add entering any subsystem,
which you invent in future. Simple and stupid. And, nevertheless, universal.

Alexey

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by serue on Mon, 06 Feb 2006 14:51:04 GMT
View Forum Message <> Reply to Message

Quoting Alexey Kuznetsov (kuznet@ms2.inr.ac.ru):
> > > 2. It is very inconvenient not to see processes inside VPS from host system.
> > > To do ps, strace, gdb etc. we have to move inside VPS. With VPID approach I can
> > > gdb even "init" process of VPS in a way invisible to VPS, see?
> >
> > that's another container model issue again. your init process of a VPS
> > could be the real init. why do you need a fake one ? just trying to
> > understand all the issues you had to solve and I'm sure they are valid.
>
> It is not a fake init, it is a real init. Main goal of openvz is to allow
> to start even f.e. the whole instance of stock redhat inside container
> including standard init. It is not a strict architectural requirement,
> but this option must be present.
>
> BTW the question sounds strange. I would think that in (container, pid)
> approach among another limitations you get requirement that you _must_
> have a child reaper inside container. With VPIDs this does not matter,
> wait() made by parent inside host always returns global pid of child.
> But with (container, pid) children can be reaped only inside container, right?

Nah, we can just special-case the lookup for pid==1 to always return the
single real init, or insert the single real init into the pidhash for
every container.

If we're going to provide the option which you need to have your own
init, then I guess the best behavior is to add a flag to whatever
mechanism creates a container specifying to either hash the real init
into the pidhash, or make the first exec()d process the container's
init? Does that seem reasonable?

thanks,
-serge

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by Alexey Kuznetsov on Mon, 06 Feb 2006 15:51:01 GMT
View Forum Message <> Reply to Message

Hello!

> into the pidhash, or make the first exec()d process the container's
> init? Does that seem reasonable?

It is exactly what openvz does, the first task becomes child reaper.

Cedric asked why do we have an init in each container, I answered that
we want to. But openvz approach is enough flexible not to do this, nothing
will break if a container process is reparented to normal init.

The question was not about openvz, it was about (container,pid) approach.
How are you going to reap chidren without a child reaper inside container?
If you reparent all the children to a single init in root container,
what does wait() return? In openvz it returns global pid and child is buried
in peace. If you do not have global pid, you cannot just return private pid.

Alexey

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by serue on Mon, 06 Feb 2006 16:24:54 GMT
View Forum Message <> Reply to Message

Quoting Alexey Kuznetsov (kuznet@ms2.inr.ac.ru):
> Hello!
>
> > into the pidhash, or make the first exec()d process the container's
> > init? Does that seem reasonable?
>
> It is exactly what openvz does, the first task becomes child reaper.
>
> Cedric asked why do we have an init in each container, I answered that
> we want to. But openvz approach is enough flexible not to do this, nothing
> will break if a container process is reparented to normal init.
>
> The question was not about openvz, it was about (container,pid) approach.
> How are you going to reap chidren without a child reaper inside container?
> If you reparent all the children to a single init in root container,
> what does wait() return? In openvz it returns global pid and child is buried
> in peace. If you do not have global pid, you cannot just return private pid.

Yes, /sbin/init would need to be modified to use whatever enhanced api
is produced to support cross-container wait and pidlookup. I'm not sure
that in this set of threads we've discussed enhanced wait requirements,
though for pidlookup it's been mentioned that we can just require some
kind of

 sys_execute_container(pid_lookup(pid));

to avoid having to expand userspace pid lookup routines - ie to avoid
having any userspace code (except init) need to know about containers.

But perhaps a per-container init is in fact cleaner. Other opinions?

-serge

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by Cedric Le Goater on Tue, 07 Feb 2006 09:15:12 GMT
View Forum Message <> Reply to Message

Bonjour!

Alexey Kuznetsov wrote:

> [...]
>
> We could force each process visiting container to daemonize and to setsid().
> But do not forget that pid space is just a little part of the whole engine,
> to force full isolation we have to close all the files opened
> in root container, to get rid of its memory space before entering container
> etc. But it makes not so much of sense, because in any case we have to keep
> at least some way to communicate to host. F.e. even when we allow to pass
> only an open pipe, we immediately encounter the situation when a file
> owned by one container could refer to processes of another container.
>
> So that, the only way to enforce full isolation is to prohibit
> "vzctl exec/enter" as whole.

containers are useful, even without migration. No doubt on that.

But, at the end, long long term probably, if we want to have a mobile
container under linux, we need to address all the issues from the start and
take them into account in the design. So, if we need to add some
constraints on the container init process (child reaper) or the resource
isolation, pid for example, to make sure a container is migratable, I think
we should start to think about it now.

By the time we reach that state, openvz would be have been rewritten a few
times already like any good software. nope ? :)

>>We've been living with the vpid approach also for years and we found issues
>>that we haven't solve at restart. So we think we might do a better job with
>>another. But, this still needs to be confirmed :)
>
> What are the issues?

The one above.

Having containers which are not migratable because their execution
environment was not contrained enough is a pity I think.

Containers are useful for isolation but being able to swsuspend them and
migrate them is even more interesting ! and fun.

> The only inconvenience which I encountered until now
> is a little problem with stray pids. F.e. this happens with flock().
> Corresponding kernel structure contains some useless (actually, illegal
> and contradicting to the nature of flock()) reference to pid.
> If the process took the lock and exited, stray pid remains forever and points
> to nowhere. In this case it is silly to prohibit checkpointing,
> but we have to restore the flock to a lock with pointing to the same point
> in the sky, i.e. to nowhere. With (container, pid) approach we would
> restore it pointing to exactly the same empty place in the sky, with
> vpids we have to choose a new place. Ugly, but not a real issue.

thanks for your insights ! I hope we will have plenty of these issues to
talk about.

c.

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by Cedric Le Goater on Tue, 07 Feb 2006 09:46:27 GMT
View Forum Message <> Reply to Message

Alexey Kuznetsov wrote:

> The question was not about openvz, it was about (container,pid) approach.
> How are you going to reap chidren without a child reaper inside container?
> If you reparent all the children to a single init in root container,
> what does wait() return? In openvz it returns global pid and child is buried
> in peace. If you do not have global pid, you cannot just return private pid.

I think the "child reaper" question is not related to the (container,pid)
approach or the vpid approach. This is another question on who is the
parent of a container and how does it behaves.

We have choosen to first follow a simple "path", complete pid isolation
being the main constraint : a container is created by exec'ing a process in
it. That first process is detached from its parent processes and becomes
child of init (already running), session leader, and process group leader.
We could eventually add a daemon to act as a init process for the container.

Now, there are other ways of seeing a container parenthood, openvz, eric,

vserver, etc. We should agree on this or find a way to have different model
cohabitate.

C.

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by dev on Tue, 07 Feb 2006 11:42:38 GMT
View Forum Message <> Reply to Message

>>The question was not about openvz, it was about (container,pid) approach.
>>How are you going to reap chidren without a child reaper inside container?
>>If you reparent all the children to a single init in root container,
>>what does wait() return? In openvz it returns global pid and child is buried
>>in peace. If you do not have global pid, you cannot just return private pid.
>
> I think the "child reaper" question is not related to the (container,pid)
> approach or the vpid approach. This is another question on who is the
> parent of a container and how does it behaves.
it is related. How reaps the last process in container when it dies?
what does waitpid() return?

> We have choosen to first follow a simple "path", complete pid isolation
> being the main constraint : a container is created by exec'ing a process in
> it.
Why to exec? It was asked already some times...

> That first process is detached from its parent processes and becomes
> child of init (already running), session leader, and process group leader.
> We could eventually add a daemon to act as a init process for the container.
See my prev question. Who reaps your init itself?

> Now, there are other ways of seeing a container parenthood, openvz, eric,
> vserver, etc. We should agree on this or find a way to have different model
> cohabitate.

Kirill

Subject: Re: [RFC][PATCH 5/7] VPIDs: vpid/pid conversion in VPID enabled case
Posted by Cedric Le Goater on Tue, 07 Feb 2006 12:59:10 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:

>> I think the "child reaper" question is not related to the (container,pid)
>> approach or the vpid approach. This is another question on who is the

>> parent of a container and how does it behaves.
>
> it is related. How reaps the last process in container when it dies?
> what does waitpid() return?

The current init sigchild handler does that very well and is also pid
friendly :)

>> We have choosen to first follow a simple "path", complete pid isolation
>> being the main constraint : a container is created by exec'ing a
>> process in it.
>
> Why to exec? It was asked already some times...

Just restarting the thread on how a container should be created ...

C.

Subject: Re: [RFC][PATCH 2/7] VPIDs: pid/vpid conversions
Posted by ebiederm on Wed, 08 Feb 2006 20:29:30 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

> This is one of the major patches,
> it adds vpid-to-pid conversions by placing macros
> introduced in diff-vpid-macro patch.
>
> Note that in CONFIG_VIRTUAL_PIDS=n case these macros expand to default code.

Do you know how incomplete this patch is?
```
 drivers/char/tty_io.c |    7 +++++--
 fs/binfmt_elf.c       |   16 ++++++++--------
 fs/exec.c             |    4 ++--
 fs/fcntl.c            |    3 ++-
 fs/locks.c            |    4 ++--
 fs/proc/array.c       |   18 ++++++++++--------
 fs/proc/base.c        |    6 +++---
 include/net/scm.h     |    2 +-
 ipc/msg.c             |    6 +++---
 ipc/sem.c             |    8 ++++----
 ipc/shm.c             |    6 +++---
 kernel/capability.c   |    8 ++++++--
 kernel/exit.c         |   28 ++++++++++++++++++++++------
 kernel/fork.c         |    2 +-
 kernel/sched.c        |    2 +-
 kernel/signal.c       |   23 +++++++++++++++--------
```

```
 kernel/sys.c         |   37 ++++++++++++++++++++++++++++++++++++----------
 kernel/timer.c       |    6 +++---
 net/core/scm.c       |    2 +-
 net/unix/af_unix.c   |    8 +++++-----
 20 files changed, 121 insertions(+), 75 deletions(-)
```

You missed drivers/char/drm, and in your shipping OpenVZ patch.
You missed get_xpid() on alpha.
You missed nfs.

All it seems you have found is the low hanging fruit where pids are used.
Without compile errors to help I don't know how you are ever going to find
everything, especially with the kernel constantly changing.

Honestly this approach looks like a maintenance nightmare, you didn't
even correctly handle all of the interfaces you posted in you patch.

I suspect the tagging of the VPIDS and the WARN_ON's help so you have
a chance of catching things if someone uses a code path you haven't
caught. But I don't see how you can possibly get full kernel
coverage.

Is there a plan to catch all of the in-kernel use of pids that I am
being to dense to see?

Eric

> Kirill

```
> --- ./kernel/capability.c.vpid_core 2006-02-02 14:15:35.152784704 +0300
> +++ ./kernel/capability.c 2006-02-02 14:33:58.808003608 +0300
> @@ -67,7 +67,7 @@ asmlinkage long sys_capget(cap_user_head
>       spin_lock(&task_capability_lock);
>       read_lock(&tasklist_lock);
>
> -     if (pid && pid != current->pid) {
> +     if (pid && pid != virt_pid(current)) {
>        target = find_task_by_pid(pid);
>        if (!target) {
>             ret = -ESRCH;
> @@ -100,6 +100,10 @@ static inline int cap_set_pg(int pgrp, k
>   int ret = -EPERM;
>   int found = 0;
>
> + pgrp = vpid_to_pid(pgrp);
> + if (pgrp < 0)
> +  return ret;
> +
```

> do_each_task_pid(pgrp, PIDTYPE_PGID, g) {
>   target = g;
>   while_each_thread(g, target) {
> @@ -199,7 +203,7 @@ asmlinkage long sys_capset(cap_user_head
>   spin_lock(&task_capability_lock);
>   read_lock(&tasklist_lock);
>
> -   if (pid > 0 && pid != current->pid) {
> +   if (pid > 0 && pid != virt_pid(current)) {
>   target = find_task_by_pid(pid);
>   if (!target) {
>   ret = -ESRCH;

You missed cap_set_all.

Subject: Re: [RFC][PATCH 2/7] VPIDs: pid/vpid conversions
Posted by Alexey Kuznetsov on Wed, 08 Feb 2006 23:53:48 GMT
View Forum Message <> Reply to Message

Hello!

> Do you know how incomplete this patch is?

The question is for me. It handles all the subsystems which are allowed
to be used inside openvz containers. And _nothing_ more, it would be pure S&M.

> You missed get_xpid() on alpha.

And probably something similar on all the archs except for i386/x86_64/ia64.

> Is there a plan to catch all of the in-kernel use of pids

grep for ->pid,->tgid,->pgid,->session and look. What could be better? :-)

> You missed cap_set_all.

No doubts, something is missing. Please, could you show how to fix it
or to point directly at the place. Thank you.

Actually, you cycled on this pid problem. If you think private pid spaces
are really necessary, it is prefectly OK. openvz (and, maybe, all VPS-oriented
solutions) do _not_ need this (well, look, virtuozzo is a mature product
for 5 years already, and vpids were added very recently for one specific

purpose), but can live within private spaces or just in peace with them.
We can even apply vpids on top on pid spaces to preserve global process tree.
Provided you leave a chance not to enforce use of private pid spaces
inside containers, of course.

Alexey

Subject: Re: [RFC][PATCH 2/7] VPIDs: pid/vpid conversions
Posted by ebiederm on Thu, 09 Feb 2006 00:37:31 GMT
View Forum Message <> Reply to Message

Alexey Kuznetsov <kuznet@ms2.inr.ac.ru> writes:

> Hello!
>
>> Do you know how incomplete this patch is?
>
> The question is for me. It handles all the subsystems which are allowed
> to be used inside openvz containers. And _nothing_ more, it would be pure S&M.

I agree and this is why I don't like VPIDS I don't see a way for them
to be anything but pure S&M.

>> Is there a plan to catch all of the in-kernel use of pids
>
> grep for ->pid,->tgid,->pgid,->session and look. What could be better? :-)

Ouch. I know there are cases that the above test fails for. Which
is why I prefer an interface that takes a global reference and gives
you a compile error if you don't. You are much more likely to catch
all of the users that way.

>> You missed cap_set_all.
>
> No doubts, something is missing. Please, could you show how to fix it
> or to point directly at the place. Thank you.

In capability.c it does for_each_thread or something like that. It is
very similar to cap_set_pg. But in a virtual context all != all :)

The current OpenVZ patch appears to at least catch cap_set_all.

> Actually, you cycled on this pid problem. If you think private pid spaces
> are really necessary, it is prefectly OK. openvz (and, maybe, all VPS-oriented
> solutions) do _not_ need this (well, look, virtuozzo is a mature product
> for 5 years already, and vpids were added very recently for one specific
> purpose), but can live within private spaces or just in peace with them.

> We can even apply vpids on top on pid spaces to preserve global process tree.
> Provided you leave a chance not to enforce use of private pid spaces
> inside containers, of course.

I think for people doing migration a private pid space in some form is
necessary, I agree it is generally overkill for the VPS case but if it
is efficient it should be usable. And certainly having facilities
like this be optional seems very important.

My problem with the vpid case and it's translate at the kernel
boundary is that boundary is huge, and there is no compile time
checking to help you find the problem users. So I don't think vpids
make a solution that can be maintained, and thus merging them looks
like a very bad idea.

Eric

Subject: Re: [RFC][PATCH 2/7] VPIDs: pid/vpid conversions
Posted by Alexey Kuznetsov on Thu, 09 Feb 2006 01:11:26 GMT
View Forum Message <> Reply to Message

Hello!

> In capability.c it does for_each_thread or something like that. It is
> very similar to cap_set_pg. But in a virtual context all != all :)

Do you mean that VPID patch does not include this? Absolutely.
VPIDs are not to limit access, the patch virtualizes pids, rather
than deals with access policy.

Take the whole openvz. Make patch -R < vpid_patch. The result is perfectly
working openvz. Only pids are not virtual, which does not matter. Capisco?

> I think for people doing migration a private pid space in some form is
> necessary,

Not "private", but "virtual". VPIDs are made only for migration, not for fun.

And word "private" is critical, f.e. for us preserving some form of pid
space is critical. It is very sad, but we cannot do anything with this,
customers will not allow to change status quo.

> My problem with the vpid case and it's translate at the kernel
> boundary is that boundary is huge

Believe me, it is surprizingly small.

Alexey

Subject: Re: [RFC][PATCH 2/7] VPIDs: pid/vpid conversions
Posted by ebiederm on Thu, 09 Feb 2006 01:36:33 GMT
View Forum Message <> Reply to Message

Alexey Kuznetsov <kuznet@ms2.inr.ac.ru> writes:

> Hello!
>
>> In capability.c it does for_each_thread or something like that. It is
>> very similar to cap_set_pg. But in a virtual context all != all :)
>
> Do you mean that VPID patch does not include this? Absolutely.
> VPIDs are not to limit access, the patch virtualizes pids, rather
> than deals with access policy.
>
> Take the whole openvz. Make patch -R < vpid_patch. The result is perfectly
> working openvz. Only pids are not virtual, which does not matter. Capisco?

Not quite.

sys_kill knows about three kinds of things referred to by pid.
- individual processes (positive pid)
- process groups (pid < -1 or pid == 0)
- The group of all processes.

When you have multiple instances of the same pid on the box
The group of all processes becomes the group of all processes I
can see. At least that was my impression.

>> I think for people doing migration a private pid space in some form is
>> necessary,
>
> Not "private", but "virtual". VPIDs are made only for migration, not for fun.
>
> And word "private" is critical, f.e. for us preserving some form of pid
> space is critical. It is very sad, but we cannot do anything with this,
> customers will not allow to change status quo.

Ok. I'm not quite certain what the difference is. In OpenVZ it appears
to be something significant. In most conversations it is not.

>> My problem with the vpid case and it's translate at the kernel
>> boundary is that boundary is huge

>
> Believe me, it is surprizingly small.

Well the number of places you need to translate may be small.
But the number of lines of code is certainly large.
Every driver, every ioctl, every other function that could
be abused ioctl.

And if the values are ever stored instead of used immediately
you can get a context mismatch.

Eric

Subject: Re: [RFC][PATCH 2/7] VPIDs: pid/vpid conversions
Posted by serue on Thu, 09 Feb 2006 02:51:30 GMT
View Forum Message <> Reply to Message

Quoting Alexey Kuznetsov (kuznet@ms2.inr.ac.ru):
> Hello!
>
> > In capability.c it does for_each_thread or something like that. It is
> > very similar to cap_set_pg. But in a virtual context all != all :)
>
> Do you mean that VPID patch does not include this? Absolutely.
> VPIDs are not to limit access, the patch virtualizes pids, rather
> than deals with access policy.
>
> Take the whole openvz. Make patch -R < vpid_patch. The result is perfectly
> working openvz. Only pids are not virtual, which does not matter. Capisco?
>
>
> > I think for people doing migration a private pid space in some form is
> > necessary,
>
> Not "private", but "virtual". VPIDs are made only for migration, not for fun.
>
> And word "private" is critical, f.e. for us preserving some form of pid
> space is critical. It is very sad, but we cannot do anything with this,

Hi,

do you mean "preserving some sort of *global* pidspace"?

If not, then I also don't understand what you're saying...

thanks,
-serge

Subject: Re: [RFC][PATCH 2/7] VPIDs: pid/vpid conversions
Posted by Alexey Kuznetsov on Thu, 09 Feb 2006 09:55:09 GMT
View Forum Message <> Reply to Message

Hello!

> do you mean "preserving some sort of *global* pidspace"?

Of course.

Alexey

Subject: Re: [RFC][PATCH 2/7] VPIDs: pid/vpid conversions
Posted by ebiederm on Thu, 09 Feb 2006 19:22:43 GMT
View Forum Message <> Reply to Message

Alexey Kuznetsov <kuznet@ms2.inr.ac.ru> writes:

> Hello!
>
>> do you mean "preserving some sort of *global* pidspace"?

> Of course.

Ok. Then my sympathies, I can understand what a difficult position
this places you in. A global pidspace resembling the one you have now
is the one idea that has been consistently shot down in all of the
discussions. So I doubt anything short of a miracle could get it
merged in the short term.

I am fairly certain that everyone who has existing management code at
this point will find it needs modifications to work with whatever
version is merged into the mainstream kernel.

Eric

Subject: Re: [RFC][PATCH 2/7] VPIDs: pid/vpid conversions
Posted by dev on Mon, 20 Feb 2006 14:55:42 GMT
View Forum Message <> Reply to Message

> Do you know how incomplete this patch is?
> You missed drivers/char/drm, and in your shipping OpenVZ patch.
> You missed get_xpid() on alpha.
> You missed nfs.
DRM/NFS code is correct.

The only correct thing you noticed is get_xpid on alpha. But this is in
fact a simple bug and half a year before we didn't care much for archs
others than i386/x86-64/ia64. That's it.

> I suspect the tagging of the VPIDS and the WARN_ON's help so you have
> a chance of catching things if someone uses a code path you haven't
> caught. But I don't see how you can possibly get full kernel
> coverage.
simple, the same way as you did, i.e. by renaming pid to tid or
something like this.

> Is there a plan to catch all of the in-kernel use of pids that I am
> being to dense to see?
if Linus will be ready to take it into mainstream, it will be caught
all. Actually only asm files should be investigated due to optimizations
similar to those on IA64/Alpha. Everything else I suppose is correct and
can be rechecked only.

And now a bit of contstructive ideas/things:

I propose to stop VPIDs discussion and switch to virtualization of
networking, IPC and so on, which is essentially the same in yours and
our solutions (openvz).

I took a look to your patch, it does actually the same things as openvz,
almost thing by thing. But it is BUGGY! You have broken IPC/networking,
many things to these subsytems are not virtualized etc. We need to
get Linus comment about which approach is the best for him, with
namespace pointers on task_struct involved by you or with effective
container pointer. It is only a matter of his taste, but the result is
effectively the same. Agree?

Actually we don't care whether virtualization introduces one container
pointer on the task struct or as you proposed many pointers to
namespaces. But you are WRONG IMHO thinking that this namespaces are
independent and this allows you more fine grained virtualization. All
these namespaces are tightly intergrated with each other(sic!).
For example, networking is coupled with sysctl, which in turn are
coupled with proc filesystem. And sysfs! You even added a piece of code
in net/core/net-sysfs.c in your patch, which is a dirty hack.
Another example, mqueues and other subsystems which use netlinks and
also depend on network context.
shmem/IPC is dependand on file system context and so on.
So it won't work when one have networking from one container and proc
from another. So I really see no much reasons to have separate
namespaces, but it is ok for me if someone really wants it this way.

We also don't care whether yours or our network virtualization will go

upstream. They do _exactly_ the same. You also virtualized IPv6 which is
good, since we have only IPv4, but you totally missed netfilters, which
is bad :) So again the only difference is that we have effective
container on the task, while you prefer to take it from sk/netdev or
bypass as an additional function argument.

So I propose the following:
1. ask Linus about the preffered approach. I prepared an email for him
with a description of approaches.
2. start from networking/netfilters/IPC which are essentially the same
in both projects and help each other.

Kirill

Subject: Re: [RFC][PATCH 2/7] VPIDs: pid/vpid conversions
Posted by Herbert Poetzl on Mon, 20 Feb 2006 16:56:54 GMT
View Forum Message <> Reply to Message

On Mon, Feb 20, 2006 at 05:57:15PM +0300, Kirill Korotaev wrote:
> >Do you know how incomplete this patch is?
> >You missed drivers/char/drm, and in your shipping OpenVZ patch.
> >You missed get_xpid() on alpha.
> >You missed nfs.
> DRM/NFS code is correct.
>
> The only correct thing you noticed is get_xpid on alpha. But this is
> in fact a simple bug and half a year before we didn't care much for
> archs others than i386/x86-64/ia64. That's it.

sidenote on that, maybe the various archs could
switch to C implementations of those 'special'
get_xpid() and friends, as I do not think they
are a) done that often (might be wrong there)
and b) recent gcc should get that right now anyway

> >I suspect the tagging of the VPIDS and the WARN_ON's help so you have
> >a chance of catching things if someone uses a code path you haven't
> >caught. But I don't see how you can possibly get full kernel
> >coverage.
> simple, the same way as you did, i.e. by renaming pid to tid or
> something like this.
>
> >Is there a plan to catch all of the in-kernel use of pids that I am
> >being to dense to see?
> if Linus will be ready to take it into mainstream, it will be
> caught all. Actually only asm files should be investigated due to
> optimizations similar to those on IA64/Alpha. Everything else I

> suppose is correct and can be rechecked only.

> And now a bit of contstructive ideas/things:

> I propose to stop VPIDs discussion and switch to virtualization of
> networking, IPC and so on, which is essentially the same in yours and
> our solutions (openvz).

> I took a look to your patch, it does actually the same things as
> openvz, almost thing by thing. But it is BUGGY! You have broken
> IPC/networking, many things to these subsytems are not virtualized
> etc. We need to get Linus comment about which approach is the best for
> him, with namespace pointers on task_struct involved by you or with
> effective container pointer. It is only a matter of his taste, but the
> result is effectively the same. Agree?

> Actually we don't care whether virtualization introduces one container
> pointer on the task struct or as you proposed many pointers to
> namespaces. But you are WRONG IMHO thinking that this namespaces are
> independent and this allows you more fine grained virtualization. All
> these namespaces are tightly intergrated with each other(sic!).

> For example, networking is coupled with sysctl, which in turn are
> coupled with proc filesystem. And sysfs! You even added a piece of code
> in net/core/net-sysfs.c in your patch, which is a dirty hack.
> Another example, mqueues and other subsystems which use netlinks and
> also depend on network context.

> shmem/IPC is dependand on file system context and so on.
> So it won't work when one have networking from one container and proc
> from another.

the question should be: which part of proc should be part
of the pid space and which not, definitely the network
stuff would _not_ be part of the pid space ...

> So I really see no much reasons to have separate namespaces,
> but it is ok for me if someone really wants it this way.

the reasons are, as I explained several times, that folks
use 'virtualization' or 'isolation' for many different
things, just because SWsoft only uses it for VPS doesn't
meant that it cannot be used for other things

just consider isolating/virtualizing the network stack,
but leaving the processes in the same pid space, how to
do that in a sane way with a single reference?

> We also don't care whether yours or our network virtualization will go
> upstream. They do _exactly_ the same. You also virtualized IPv6 which is
> good, since we have only IPv4, but you totally missed netfilters, which
> is bad :) So again the only difference is that we have effective
> container on the task, while you prefer to take it from sk/netdev or
> bypass as an additional function argument.
>
> So I propose the following:
> 1. ask Linus about the preffered approach. I prepared an email for him
> with a description of approaches.

why do you propose, if you already did? :)

> 2. start from networking/netfilters/IPC which are essentially the same
> in both projects and help each other.

no problem with that, once Eric got there ...

best,
Herbert

> Kirill

Subject: Re: [RFC][PATCH 2/7] VPIDs: pid/vpid conversions
Posted by dev on Tue, 21 Feb 2006 16:17:59 GMT
View Forum Message <> Reply to Message

>>The only correct thing you noticed is get_xpid on alpha. But this is
>>in fact a simple bug and half a year before we didn't care much for
>>archs others than i386/x86-64/ia64. That's it.
> sidenote on that, maybe the various archs could
> switch to C implementations of those 'special'
> get_xpid() and friends, as I do not think they
> are a) done that often (might be wrong there)
> and b) recent gcc should get that right now anyway
I also wonder why it was required and can't be done in normal way...
Maybe worth trying to switch to C, really.

>>For example, networking is coupled with sysctl, which in turn are
>>coupled with proc filesystem. And sysfs! You even added a piece of code
>>in net/core/net-sysfs.c in your patch, which is a dirty hack.
>>Another example, mqueues and other subsystems which use netlinks and
>>also depend on network context.
>>shmem/IPC is dependand on file system context and so on.
>>So it won't work when one have networking from one container and proc
>>from another.
> the question should be: which part of proc should be part

> of the pid space and which not, definitely the network
> stuff would _not_ be part of the pid space ...
Ok, just one simple question:
how do you propose to handle network sysctls and network
statistics/information in proc?
how can you imagine this namespaces should work?
I see no elegant solution for this, do you? If there is any, I will be
happy with namespaces again.

>>So I really see no much reasons to have separate namespaces,
>>but it is ok for me if someone really wants it this way.
> the reasons are, as I explained several times, that folks
> use 'virtualization' or 'isolation' for many different
> things, just because SWsoft only uses it for VPS doesn't
> meant that it cannot be used for other things
Out of curiosity, do you have any _working_ examples of other usages?
I see only theoretical examples from you, but would like to hear from
anyone who _uses_/_knows_ how to use it.

> just consider isolating/virtualizing the network stack,
> but leaving the processes in the same pid space, how to
> do that in a sane way with a single reference?
I see... Any idea why this can be required?
(without proc? :))
BTW, if you have virtualized networking, but not isolated fs namespace
in this case, how are you going to handle unix sockets? Or maybe it's
another separate namespace?

>>1. ask Linus about the preffered approach. I prepared an email for him
>>with a description of approaches.
> why do you propose, if you already did? :)
because, the question was quite simple, isn't it?

>>2. start from networking/netfilters/IPC which are essentially the same
>>in both projects and help each other.
> no problem with that, once Eric got there ...

Kirill

Subject: Re: [RFC][PATCH 2/7] VPIDs: pid/vpid conversions
Posted by Herbert Poetzl on Tue, 21 Feb 2006 23:17:29 GMT
View Forum Message <> Reply to Message

On Tue, Feb 21, 2006 at 07:19:01PM +0300, Kirill Korotaev wrote:
>>>The only correct thing you noticed is get_xpid on alpha. But this is
>>>in fact a simple bug and half a year before we didn't care much for
>>>archs others than i386/x86-64/ia64. That's it.

>>sidenote on that, maybe the various archs could
>>switch to C implementations of those 'special'
>>get_xpid() and friends, as I do not think they
>>are a) done that often (might be wrong there)
>>and b) recent gcc should get that right now anyway
>I also wonder why it was required and can't be done in normal way...
>Maybe worth trying to switch to C, really.
definitely
>>>For example, networking is coupled with sysctl, which in turn are
>>>coupled with proc filesystem. And sysfs! You even added a piece of code
>>>in net/core/net-sysfs.c in your patch, which is a dirty hack.
>>>Another example, mqueues and other subsystems which use netlinks and
>>>also depend on network context.
>>>shmem/IPC is dependand on file system context and so on.
>>>So it won't work when one have networking from one container and proc
>>>from another.
>>the question should be: which part of proc should be part
>>of the pid space and which not, definitely the network
>>stuff would _not_ be part of the pid space ...
>Ok, just one simple question:
>how do you propose to handle network sysctls and network
>statistics/information in proc?
well, procfs is called procfs because it is/was?
supposed to contain process information, otherwise
it would have been called netfs or statfs or even
junkfs :)
>_how_ can you imagine this namespaces should work?
>I see no elegant solution for this, do you?
>If there is any, I will be happy with namespaces again.
junkfs parts need to be properly virtualized, the
procfs parts do not.
>>>So I really see no much reasons to have separate namespaces,
>>>but it is ok for me if someone really wants it this way.
>>the reasons are, as I explained several times, that folks
>>use 'virtualization' or 'isolation' for many different
>>things, just because SWsoft only uses it for VPS doesn't
>>meant that it cannot be used for other things
>Out of curiosity, do you have any _working_ examples of other usages?
>I see only theoretical examples from you, but would like to hear from
>anyone who _uses_/_knows_ how to use it.
seems we are going in circles here, I already gave
a detailed list of _actual_ uses which are different
from the VPS approach
>>just consider isolating/virtualizing the network stack,
>>but leaving the processes in the same pid space, how to
>>do that in a sane way with a single reference?
>I see... Any idea why this can be required?
>(without proc? :))

>BTW, if you have virtualized networking, but not isolated fs namespace
>in this case, how are you going to handle unix sockets? Or maybe it's
>another separate namespace?
two httpd servers could easily bind to a subset of
the host IP addresses while sharing the pid space
(and other spaces). guess what, that actually works
and is in use ...
>>>1. ask Linus about the preffered approach. I prepared an email for him
>>>with a description of approaches.
>>why do you propose, if you already did? :)
>because, the question was quite simple, isn't it?
no comment
>>>2. start from networking/netfilters/IPC which are essentially the same
>>>in both projects and help each other.
>>no problem with that, once Eric got there ...
>Kirill
best,
Herbert
PS: as one can see, I gave up on fixing your unreadable
quoting, so don't expect readability ...
