Subject: [PATCH 0/12] Schedule find_task_by_pid() for removal
Posted by Pavel Emelianov on Tue, 29 Jan 2008 13:38:36 GMT
View Forum Message <> Reply to Message

The function in question finds the task by its pid_t in an
initial pid namespace (init_pid_ns). This is no longer safe to
simply call it, since the current task may be in a pid namespace
and it may return the wrong task.

The proper behavior is to call the find_task_by_vpid() if the
caller is sure, that the target task lives in the same namespace
as he is, or the find_task_by_pid_ns() and specify the namespace
to find a task in.

Since the find_task_by_pid() is a well-known API call, and its
semantics changes, I think that it's better to mark it as
deprecated to warn people, that the result may differ from what
they expect and force them to use proper call.

Another case is to store the pointer to the struct pid of a desired
task and later call the pid_task() to get the task itself.

So this set just fixes the existing users of find_task_by_pid().
The only one will be left - the UML mconsole, but I'm completely
lost in this code, finding out what kind of pid is passed in
the mconsole_stack() call :(

Singed-off-by: Pavel Emelyanov <xemul@openvz.org>

Subject: [PATCH 1/12] Use find_task_by_vpid in posix timers
Posted by Pavel Emelianov on Tue, 29 Jan 2008 13:41:20 GMT
View Forum Message <> Reply to Message

All the functions that need to lookup a task by pid in posix
timers obtain this pid from a user space, and thus this value
refers to a task in the same namespace, as the current task
lives in.

So the proper behavior is to call find_task_by_vpid() here.

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

---
 kernel/posix-cpu-timers.c | 8 ++++----
 kernel/posix-timers.c | 2 +-
 2 files changed, 5 insertions(+), 5 deletions(-)

```
diff --git a/kernel/posix-cpu-timers.c b/kernel/posix-cpu-timers.c
index 0b7c82a..2eae91f 100644
--- a/kernel/posix-cpu-timers.c
+++ b/kernel/posix-cpu-timers.c
@@ -20,7 +20,7 @@ static int check_clock(const clockid_t which_clock)
   return 0;

  read_lock(&tasklist_lock);
- p = find_task_by_pid(pid);
+ p = find_task_by_vpid(pid);
  if (!p || !(CPUCLOCK_PERTHREAD(which_clock) ?
      same_thread_group(p, current) : thread_group_leader(p))) {
   error = -EINVAL;
@@ -305,7 +305,7 @@ int posix_cpu_clock_get(const clockid_t which_clock, struct timespec *tp)
    */
   struct task_struct *p;
   rcu_read_lock();
-  p = find_task_by_pid(pid);
+  p = find_task_by_vpid(pid);
   if (p) {
    if (CPUCLOCK_PERTHREAD(which_clock)) {
     if (same_thread_group(p, current)) {
@@ -354,7 +354,7 @@ int posix_cpu_timer_create(struct k_itimer *new_timer)
   if (pid == 0) {
    p = current;
   } else {
-   p = find_task_by_pid(pid);
+   p = find_task_by_vpid(pid);
    if (p && !same_thread_group(p, current))
     p = NULL;
   }
@@ -362,7 +362,7 @@ int posix_cpu_timer_create(struct k_itimer *new_timer)
   if (pid == 0) {
    p = current->group_leader;
   } else {
-   p = find_task_by_pid(pid);
+   p = find_task_by_vpid(pid);
    if (p && !thread_group_leader(p))
     p = NULL;
   }
diff --git a/kernel/posix-timers.c b/kernel/posix-timers.c
index 80d5620..c75d4e2 100644
--- a/kernel/posix-timers.c
+++ b/kernel/posix-timers.c
@@ -404,7 +404,7 @@ static struct task_struct * good_sigevent(sigevent_t * event)
  struct task_struct *rtn = current->group_leader;

  if ((event->sigev_notify & SIGEV_THREAD_ID ) &&
```

```
-  (!(rtn = find_task_by_pid(event->sigev_notify_thread_id)) ||
+  (!(rtn = find_task_by_vpid(event->sigev_notify_thread_id)) ||
     !same_thread_group(rtn, current) ||
     (event->sigev_notify & ~SIGEV_THREAD_ID) != SIGEV_SIGNAL))
   return NULL;
--
1.5.3.4
```

Subject: [PATCH 2/12] Use find_task_by_vpid in autid code
Posted by Pavel Emelianov on Tue, 29 Jan 2008 13:43:25 GMT
View Forum Message <> Reply to Message

The pid to lookup a task by is passed inside audit code
via netlink message.

Thanks to Denis Lunev, netlink packets are now (since 2.6.24)
always processed in the context of the sending task. So this
is correct to lookup the task with find_task_by_vpid() here.

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

```
---
 kernel/audit.c |    6 +++---
 1 files changed, 3 insertions(+), 3 deletions(-)

diff --git a/kernel/audit.c b/kernel/audit.c
index ae58d87..ebb9d60 100644
--- a/kernel/audit.c
+++ b/kernel/audit.c
@@ -378,7 +378,7 @@ static int audit_prepare_user_tty(pid_t pid, uid_t loginuid)
  int err;

  read_lock(&tasklist_lock);
- tsk = find_task_by_pid(pid);
+ tsk = find_task_by_vpid(pid);
  err = -ESRCH;
  if (!tsk)
   goto out;
@@ -764,7 +764,7 @@ static int audit_receive_msg(struct sk_buff *skb, struct nlmsghdr *nlh)
   struct task_struct *tsk;

   read_lock(&tasklist_lock);
-  tsk = find_task_by_pid(pid);
+  tsk = find_task_by_vpid(pid);
   if (!tsk)
    err = -ESRCH;
   else {
```

```
@@ -787,7 +787,7 @@ static int audit_receive_msg(struct sk_buff *skb, struct nlmsghdr *nlh)
   if (s->enabled != 0 && s->enabled != 1)
    return -EINVAL;
   read_lock(&tasklist_lock);
-  tsk = find_task_by_pid(pid);
+  tsk = find_task_by_vpid(pid);
   if (!tsk)
    err = -ESRCH;
   else {
--
1.5.3.4
```

Subject: [PATCH 3/12] Use find_task_by_vpid in taskstats
Posted by Pavel Emelianov on Tue, 29 Jan 2008 13:46:15 GMT
View Forum Message <> Reply to Message

The pid to lookup a task by is passed inside taskstats code
via genetlink message.

Since netlink packets are now processed in the context of the
sending task, this is correct to lookup the task with
find_task_by_vpid() here.

Besides, I fix the call to fill_pid() from taskstats_exit(),
since the tsk->pid is not required in fill_pid() in this case,
and the pid field on task_struct is going to be deprecated
as well.

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

```
---
 kernel/taskstats.c |    6 +++---
 1 files changed, 3 insertions(+), 3 deletions(-)

diff --git a/kernel/taskstats.c b/kernel/taskstats.c
index 07e86a8..4a23517 100644
--- a/kernel/taskstats.c
+++ b/kernel/taskstats.c
@@ -183,7 +183,7 @@ static int fill_pid(pid_t pid, struct task_struct *tsk,

  if (!tsk) {
   rcu_read_lock();
-  tsk = find_task_by_pid(pid);
+  tsk = find_task_by_vpid(pid);
   if (tsk)
    get_task_struct(tsk);
   rcu_read_unlock();
```

```
@@ -230,7 +230,7 @@ static int fill_tgid(pid_t tgid, struct task_struct *first,
   */
  rcu_read_lock();
  if (!first)
-  first = find_task_by_pid(tgid);
+  first = find_task_by_vpid(tgid);

  if (!first || !lock_task_sighand(first, &flags))
   goto out;
@@ -547,7 +547,7 @@ void taskstats_exit(struct task_struct *tsk, int group_dead)
  if (!stats)
   goto err;

- rc = fill_pid(tsk->pid, tsk, stats);
+ rc = fill_pid(-1, tsk, stats);
  if (rc < 0)
   goto err;

--
1.5.3.4
```

Subject: [PATCH 4/12] Don't operate with pid_t in rtmutex tester
Posted by Pavel Emelianov on Tue, 29 Jan 2008 13:49:27 GMT
View Forum Message <> Reply to Message

The proper behavior to store task's pid and get this task
later is to get the struct pid pointer and get the task
with the pid_task() call.

Make it for rt_mutex_waiter->deadlock_task_pid field.

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

```
---
 kernel/rtmutex-debug.c  |   12 ++++++++---
 kernel/rtmutex_common.h |    2 +-
 2 files changed, 10 insertions(+), 4 deletions(-)

diff --git a/kernel/rtmutex-debug.c b/kernel/rtmutex-debug.c
index 56d73cb..40ec85c 100644
--- a/kernel/rtmutex-debug.c
+++ b/kernel/rtmutex-debug.c
@@ -130,7 +130,7 @@ void debug_rt_mutex_deadlock(int detect, struct rt_mutex_waiter
*act_waiter,

  task = rt_mutex_owner(act_waiter->lock);
  if (task && task != current) {
```

```
-  act_waiter->deadlock_task_pid = task->pid;
+  act_waiter->deadlock_task_pid = get_pid(task_pid(task));
   act_waiter->deadlock_lock = lock;
  }
 }
@@ -142,9 +142,12 @@ void debug_rt_mutex_print_deadlock(struct rt_mutex_waiter *waiter)
  if (!waiter->deadlock_lock || !rt_trace_on)
   return;

- task = find_task_by_pid(waiter->deadlock_task_pid);
- if (!task)
+ rcu_read_lock();
+ task = pid_task(waiter->deadlock_task_pid, PIDTYPE_PID);
+ if (!task) {
+  rcu_read_unlock();
   return;
+ }

  TRACE_OFF_NOLOCK();

@@ -173,6 +176,7 @@ void debug_rt_mutex_print_deadlock(struct rt_mutex_waiter *waiter)
   current->comm, task_pid_nr(current));
  dump_stack();
  debug_show_all_locks();
+ rcu_read_unlock();

  printk("[ turning off deadlock detection."
         "Please report this trace. ]\n\n");
@@ -203,10 +207,12 @@ void debug_rt_mutex_init_waiter(struct rt_mutex_waiter *waiter)
  memset(waiter, 0x11, sizeof(*waiter));
  plist_node_init(&waiter->list_entry, MAX_PRIO);
  plist_node_init(&waiter->pi_list_entry, MAX_PRIO);
+ waiter->deadlock_task_pid = NULL;
 }

 void debug_rt_mutex_free_waiter(struct rt_mutex_waiter *waiter)
 {
+ put_pid(waiter->deadlock_task_pid);
  TRACE_WARN_ON(!plist_node_empty(&waiter->list_entry));
  TRACE_WARN_ON(!plist_node_empty(&waiter->pi_list_entry));
  TRACE_WARN_ON(waiter->task);
diff --git a/kernel/rtmutex_common.h b/kernel/rtmutex_common.h
index 2d3b835..e124bf5 100644
--- a/kernel/rtmutex_common.h
+++ b/kernel/rtmutex_common.h
@@ -51,7 +51,7 @@ struct rt_mutex_waiter {
  struct rt_mutex  *lock;
 #ifdef CONFIG_DEBUG_RT_MUTEXES
```

```
  unsigned long  ip;
- pid_t   deadlock_task_pid;
+ struct pid  *deadlock_task_pid;
  struct rt_mutex  *deadlock_lock;
 #endif
 };
--
1.5.3.4
```

Subject: [PATCH 5/12] Handle pid namespaces in cgroups code
Posted by Pavel Emelianov on Tue, 29 Jan 2008 13:52:25 GMT
View Forum Message <> Reply to Message

There's one place that works with task pids - its the "tasks" file
in cgroups. The read/write handlers assume, that the pid values
go to/come from the user space and thus it is a virtual pid, i.e.
the pid as it is seen from inside a namespace.

Tune the code accordingly.

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

```
---
 kernel/cgroup.c |    4 ++--
 1 files changed, 2 insertions(+), 2 deletions(-)

diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index 2c5cccb..4766bb6 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -1269,7 +1269,7 @@ static int attach_task_by_pid(struct cgroup *cgrp, char *pidbuf)

  if (pid) {
   rcu_read_lock();
-  tsk = find_task_by_pid(pid);
+  tsk = find_task_by_vpid(pid);
   if (!tsk || tsk->flags & PF_EXITING) {
    rcu_read_unlock();
    return -ESRCH;
@@ -1955,7 +1955,7 @@ static int pid_array_load(pid_t *pidarray, int npids, struct cgroup *cgrp)
  while ((tsk = cgroup_iter_next(cgrp, &it))) {
   if (unlikely(n == npids))
    break;
-  pidarray[n++] = task_pid_nr(tsk);
+  pidarray[n++] = task_pid_vnr(tsk);
  }
  cgroup_iter_end(cgrp, &it);
```

```
  return n;
--
1.5.3.4
```

Subject: [PATCH 6/12] gfs2: make gfs2_glock.gl_owner_pid be a struct pid *
Posted by Pavel Emelianov on Tue, 29 Jan 2008 13:55:47 GMT
View Forum Message <> Reply to Message

The gl_owner_pid field is used to get the lock owning
task by its pid, so make it in a proper manner, i.e.
by using the struct pid pointer and pid_task() function.

The pid_task() becomes exported for the gfs2 module.

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

```
---
 fs/gfs2/glock.c  |   19 ++++++++++++-------
 fs/gfs2/incore.h |    2 +-
 2 files changed, 13 insertions(+), 8 deletions(-)

diff --git a/fs/gfs2/glock.c b/fs/gfs2/glock.c
index 80e09c5..5fe585f 100644
--- a/fs/gfs2/glock.c
+++ b/fs/gfs2/glock.c
@@ -334,7 +334,7 @@ int gfs2_glock_get(struct gfs2_sbd *sdp, u64 number,
  gl->gl_state = LM_ST_UNLOCKED;
  gl->gl_demote_state = LM_ST_EXCLUSIVE;
  gl->gl_hash = hash;
- gl->gl_owner_pid = 0;
+ gl->gl_owner_pid = NULL;
  gl->gl_ip = 0;
  gl->gl_ops = glops;
  gl->gl_req_gh = NULL;
@@ -631,7 +631,7 @@ static void gfs2_glmutex_lock(struct gfs2_glock *gl)
   wait_on_holder(&gh);
   gfs2_holder_uninit(&gh);
  } else {
-  gl->gl_owner_pid = current->pid;
+  gl->gl_owner_pid = get_pid(task_pid(current));
   gl->gl_ip = (unsigned long)__builtin_return_address(0);
   spin_unlock(&gl->gl_spin);
  }
@@ -652,7 +652,7 @@ static int gfs2_glmutex_trylock(struct gfs2_glock *gl)
  if (test_and_set_bit(GLF_LOCK, &gl->gl_flags)) {
   acquired = 0;
  } else {
```

```
-  gl->gl_owner_pid = current->pid;
+  gl->gl_owner_pid = get_pid(task_pid(current));
   gl->gl_ip = (unsigned long)__builtin_return_address(0);
  }
  spin_unlock(&gl->gl_spin);
@@ -668,12 +668,17 @@ static int gfs2_glmutex_trylock(struct gfs2_glock *gl)

 static void gfs2_glmutex_unlock(struct gfs2_glock *gl)
 {
+ struct pid *pid;
+
  spin_lock(&gl->gl_spin);
  clear_bit(GLF_LOCK, &gl->gl_flags);
- gl->gl_owner_pid = 0;
+ pid = gl->gl_owner_pid;
+ gl->gl_owner_pid = NULL;
  gl->gl_ip = 0;
  run_queue(gl);
  spin_unlock(&gl->gl_spin);
+
+ put_pid(pid);
 }

 /**
@@ -1877,13 +1882,13 @@ static int dump_glock(struct glock_iter *gi, struct gfs2_glock *gl)
  print_dbg(gi, "  gl_ref = %d\n", atomic_read(&gl->gl_ref));
  print_dbg(gi, "  gl_state = %u\n", gl->gl_state);
  if (gl->gl_owner_pid) {
-  gl_owner = find_task_by_pid(gl->gl_owner_pid);
+  gl_owner = pid_task(gl->gl_owner_pid, PIDTYPE_PID);
   if (gl_owner)
    print_dbg(gi, "  gl_owner = pid %d (%s)\n",
-      gl->gl_owner_pid, gl_owner->comm);
+      pid_nr(gl->gl_owner_pid), gl_owner->comm);
   else
    print_dbg(gi, "  gl_owner = %d (ended)\n",
-      gl->gl_owner_pid);
+      pid_nr(gl->gl_owner_pid));
  } else
   print_dbg(gi, "  gl_owner = -1\n");
  print_dbg(gi, "  gl_ip = %lu\n", gl->gl_ip);
diff --git a/fs/gfs2/incore.h b/fs/gfs2/incore.h
index 1339996..8ad1c3f 100644
--- a/fs/gfs2/incore.h
+++ b/fs/gfs2/incore.h
@@ -182,7 +182,7 @@ struct gfs2_glock {
  unsigned int gl_hash;
  unsigned int gl_demote_state; /* state requested by remote node */
```

```
  unsigned long gl_demote_time; /* time of first demote request */
- pid_t gl_owner_pid;
+ struct pid *gl_owner_pid;
  unsigned long gl_ip;
  struct list_head gl_holders;
  struct list_head gl_waiters1; /* HIF_MUTEX */
diff --git a/kernel/pid.c b/kernel/pid.c
index e4e5fc8..4776915 100644
--- a/kernel/pid.c
+++ b/kernel/pid.c
@@ -367,6 +367,7 @@ struct task_struct *pid_task(struct pid *pid, enum pid_type type)
  }
  return result;
 }
+EXPORT_SYMBOL(pid_task);

 /*
  * Must be called under rcu_read_lock() or with tasklist_lock read-held.
--
1.5.3.4
```

Subject: [PATCH 7/12] gfs2: make gfs2_holder.gh_owner_pid be a struct pid *
Posted by Pavel Emelianov on Tue, 29 Jan 2008 13:57:41 GMT
View Forum Message <> Reply to Message

The gl_owner_pid field is used to get the holder task by its
pid and check whether the current is a holder, so make it in a
proper manner, i.e. via the struct pid * manipulations.

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

```
---
 fs/gfs2/glock.c  |   18 ++++++++++-------
 fs/gfs2/glock.h  |    4 +++-
 fs/gfs2/incore.h |    2 +-
 3 files changed, 15 insertions(+), 9 deletions(-)

diff --git a/fs/gfs2/glock.c b/fs/gfs2/glock.c
index 5fe585f..5a2b625 100644
--- a/fs/gfs2/glock.c
+++ b/fs/gfs2/glock.c
@@ -399,7 +399,7 @@ void gfs2_holder_init(struct gfs2_glock *gl, unsigned int state, unsigned
flags,
  INIT_LIST_HEAD(&gh->gh_list);
  gh->gh_gl = gl;
  gh->gh_ip = (unsigned long)__builtin_return_address(0);
- gh->gh_owner_pid = current->pid;
```

```
+ gh->gh_owner_pid = get_pid(task_pid(current));
  gh->gh_state = state;
  gh->gh_flags = flags;
  gh->gh_error = 0;
@@ -433,6 +433,7 @@ void gfs2_holder_reinit(unsigned int state, unsigned flags, struct
gfs2_holder *

 void gfs2_holder_uninit(struct gfs2_holder *gh)
 {
+ put_pid(gh->gh_owner_pid);
  gfs2_glock_put(gh->gh_gl);
  gh->gh_gl = NULL;
  gh->gh_ip = 0;
@@ -1050,7 +1051,7 @@ static int glock_wait_internal(struct gfs2_holder *gh)
 }

 static inline struct gfs2_holder *
-find_holder_by_owner(struct list_head *head, pid_t pid)
+find_holder_by_owner(struct list_head *head, struct pid *pid)
 {
  struct gfs2_holder *gh;

@@ -1087,7 +1088,7 @@ static void add_to_queue(struct gfs2_holder *gh)
  struct gfs2_glock *gl = gh->gh_gl;
  struct gfs2_holder *existing;

- BUG_ON(!gh->gh_owner_pid);
+ BUG_ON(gh->gh_owner_pid == NULL);
  if (test_and_set_bit(HIF_WAIT, &gh->gh_iflags))
   BUG();

@@ -1097,12 +1098,14 @@ static void add_to_queue(struct gfs2_holder *gh)
   if (existing) {
    print_symbol(KERN_WARNING "original: %s\n",
         existing->gh_ip);
-   printk(KERN_INFO "pid : %d\n", existing->gh_owner_pid);
+   printk(KERN_INFO "pid : %d\n",
+     pid_nr(existing->gh_owner_pid));
    printk(KERN_INFO "lock type : %d lock state : %d\n",
          existing->gh_gl->gl_name.ln_type,
          existing->gh_gl->gl_state);
    print_symbol(KERN_WARNING "new: %s\n", gh->gh_ip);
-   printk(KERN_INFO "pid : %d\n", gh->gh_owner_pid);
+   printk(KERN_INFO "pid : %d\n",
+     pid_nr(gh->gh_owner_pid));
    printk(KERN_INFO "lock type : %d lock state : %d\n",
          gl->gl_name.ln_type, gl->gl_state);
    BUG();
```

```
@@ -1803,8 +1806,9 @@ static int dump_holder(struct glock_iter *gi, char *str,

  print_dbg(gi, "  %s\n", str);
  if (gh->gh_owner_pid) {
-  print_dbg(gi, "    owner = %ld ", (long)gh->gh_owner_pid);
-  gh_owner = find_task_by_pid(gh->gh_owner_pid);
+  print_dbg(gi, "    owner = %ld ",
+    (long)pid_nr(gh->gh_owner_pid));
+  gh_owner = pid_task(gh->gh_owner_pid, PIDTYPE_PID);
    if (gh_owner)
     print_dbg(gi, "(%s)\n", gh_owner->comm);
    else
diff --git a/fs/gfs2/glock.h b/fs/gfs2/glock.h
index b16f604..2f9c6d1 100644
--- a/fs/gfs2/glock.h
+++ b/fs/gfs2/glock.h
@@ -36,11 +36,13 @@ static inline int gfs2_glock_is_locked_by_me(struct gfs2_glock *gl)
 {
  struct gfs2_holder *gh;
  int locked = 0;
+ struct pid *pid;

  /* Look in glock's list of holders for one with current task as owner */
  spin_lock(&gl->gl_spin);
+ pid = task_pid(current);
  list_for_each_entry(gh, &gl->gl_holders, gh_list) {
-  if (gh->gh_owner_pid == current->pid) {
+  if (gh->gh_owner_pid == pid) {
     locked = 1;
     break;
    }
diff --git a/fs/gfs2/incore.h b/fs/gfs2/incore.h
index 8ad1c3f..2a8d810 100644
--- a/fs/gfs2/incore.h
+++ b/fs/gfs2/incore.h
@@ -151,7 +151,7 @@ struct gfs2_holder {
  struct list_head gh_list;

  struct gfs2_glock *gh_gl;
- pid_t gh_owner_pid;
+ struct pid *gh_owner_pid;
  unsigned int gh_state;
  unsigned gh_flags;

--
1.5.3.4
```

Subject: [PATCH 8/12] frv: use find_task_by_vpid in cxn_pin_by_pid
Posted by Pavel Emelianov on Tue, 29 Jan 2008 13:59:49 GMT
View Forum Message <> Reply to Message

The function is question gets the pid from sysctl table,
so this one is a virtual pid, i.e. the pid of a task as it
is seen from inside a namespace.

So the find_task_by_vpid() must be used here.

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

---
 arch/frv/mm/mmu-context.c | 2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)

diff --git a/arch/frv/mm/mmu-context.c b/arch/frv/mm/mmu-context.c
index 1530a41..81757d5 100644
--- a/arch/frv/mm/mmu-context.c
+++ b/arch/frv/mm/mmu-context.c
@@ -181,7 +181,7 @@ int cxn_pin_by_pid(pid_t pid)

  /* get a handle on the mm_struct */
  read_lock(&tasklist_lock);
- tsk = find_task_by_pid(pid);
+ tsk = find_task_by_vpid(pid);
  if (tsk) {
   ret = -EINVAL;

--
1.5.3.4

Subject: [PATCH 9/12] ia64: make pfm_get_task work with virtual pids
Posted by Pavel Emelianov on Tue, 29 Jan 2008 14:02:03 GMT
View Forum Message <> Reply to Message

This pid comes from user space, so treat it accordingly.

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

---
 arch/ia64/kernel/perfmon.c | 4 ++--
 1 files changed, 2 insertions(+), 2 deletions(-)

diff --git a/arch/ia64/kernel/perfmon.c b/arch/ia64/kernel/perfmon.c
index 73e7c2e..da1cff0 100644
--- a/arch/ia64/kernel/perfmon.c

```
+++ b/arch/ia64/kernel/perfmon.c
@@ -2654,11 +2654,11 @@ pfm_get_task(pfm_context_t *ctx, pid_t pid, struct task_struct
**task)
  /* XXX: need to add more checks here */
  if (pid < 2) return -EPERM;

- if (pid != current->pid) {
+ if (pid != task_pid_vnr(current)) {

   read_lock(&tasklist_lock);

-  p = find_task_by_pid(pid);
+  p = find_task_by_vpid(pid);

   /* make sure task cannot go away while we operate on it */
   if (p) get_task_struct(p);
--
1.5.3.4
```

Subject: [PATCH 10/12] ia64: fix ptrace inside a namespace
Posted by Pavel Emelianov on Tue, 29 Jan 2008 14:04:20 GMT
View Forum Message <> Reply to Message

The ia64 has its own sys_ptrace implementation and it was
overlooked when the pid namespaces patches were sent.

Use find_task_by_vpid() in it.

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

```
---
 arch/ia64/kernel/ptrace.c |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)

diff --git a/arch/ia64/kernel/ptrace.c b/arch/ia64/kernel/ptrace.c
index 2e96f17..a671ef8 100644
--- a/arch/ia64/kernel/ptrace.c
+++ b/arch/ia64/kernel/ptrace.c
@@ -1437,7 +1437,7 @@ sys_ptrace (long request, pid_t pid, unsigned long addr, unsigned long
data)
  ret = -ESRCH;
  read_lock(&tasklist_lock);
  {
-  child = find_task_by_pid(pid);
+  child = find_task_by_vpid(pid);
   if (child) {
    if (peek_or_poke)
```

```
     child = find_thread_for_addr(child, addr);
--
1.5.3.4
```

Subject: [PATCH 11/12] mips: use find_task_by_vpid in system calls
Posted by Pavel Emelianov on Tue, 29 Jan 2008 14:06:32 GMT
View Forum Message <> Reply to Message

There are some places left in mips, that lookup task in
initial namespace, while the code doing so gets the pid
from the user space and thus must treat it as virtual.

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

```
---
 arch/mips/kernel/mips-mt-fpaff.c |    2 +-
 arch/mips/kernel/sysirix.c       |    2 +-
 2 files changed, 2 insertions(+), 2 deletions(-)

diff --git a/arch/mips/kernel/mips-mt-fpaff.c b/arch/mips/kernel/mips-mt-fpaff.c
index bb4f00c..df4d3f2 100644
--- a/arch/mips/kernel/mips-mt-fpaff.c
+++ b/arch/mips/kernel/mips-mt-fpaff.c
@@ -36,7 +36,7 @@ unsigned long mt_fpemul_threshold = 0;
  */
 static inline struct task_struct *find_process_by_pid(pid_t pid)
 {
- return pid ? find_task_by_pid(pid) : current;
+ return pid ? find_task_by_vpid(pid) : current;
 }


diff --git a/arch/mips/kernel/sysirix.c b/arch/mips/kernel/sysirix.c
index 672fba8..c357762 100644
--- a/arch/mips/kernel/sysirix.c
+++ b/arch/mips/kernel/sysirix.c
@@ -111,7 +111,7 @@ asmlinkage int irix_prctl(unsigned option, ...)
   printk("irix_prctl[%s:%d]: Wants PR_ISBLOCKED\n",
          current->comm, current->pid);
   read_lock(&tasklist_lock);
-  task = find_task_by_pid(va_arg(args, pid_t));
+  task = find_task_by_vpid(va_arg(args, pid_t));
   error = -ESRCH;
   if (error)
    error = (task->run_list.next != NULL);
--
1.5.3.4
```

Subject: [PATCH 12/12] Deprecate the find_task_by_pid
Posted by Pavel Emelianov on Tue, 29 Jan 2008 14:09:55 GMT
View Forum Message <> Reply to Message

There are some places that are known to operate on tasks'
global pids only:

* the rest_init() call (called on boot)
* the kgdb's getthread
* the create_kthread() (since the kthread is run in init ns)

So use the find_task_by_pid_ns(..., &init_pid_ns) there
and schedule the find_task_by_pid for removal.

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

 Documentation/feature-removal-schedule.txt | 19 ++++++++++++++++++
 include/linux/sched.h | 2 +-
 init/main.c | 2 +-
 kernel/kgdb.c | 2 +-
 kernel/kthread.c | 2 +-
 5 files changed, 23 insertions(+), 4 deletions(-)

diff --git a/Documentation/feature-removal-schedule.txt
b/Documentation/feature-removal-schedule.txt
index be6c2db..1e6960d 100644
--- a/Documentation/feature-removal-schedule.txt
+++ b/Documentation/feature-removal-schedule.txt
@@ -264,3 +264,22 @@ Why: These drivers are superseded by i810fb, intelfb and savagefb.
 Who: Jean Delvare <khali@linux-fr.org>

+
+What: find_task_by_pid
+When: 2.6.26
+Why: With pid namespaces, calling this funciton will return the
+ wrong task when called from inside a namespace.
+
+ The best way to save a task pid and find a task by this
+ pid later, is to find this task's struct pid pointer (or get
+ it directly from the task) and call pid_task() later.
+
+ If someone really needs to get a task by its pid_t, then
+ he most likely needs the find_task_by_vpid() to get the
+ task from the same namespace as the current task is in, but
+ this may be not so in general.
+
+Who: Pavel Emelyanov <xemul@openvz.org>

```
+
+-------------------------
+
diff --git a/include/linux/sched.h b/include/linux/sched.h
index 198659b..ccb9bba 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1617,7 +1617,7 @@ extern struct pid_namespace init_pid_ns;
 extern struct task_struct *find_task_by_pid_type_ns(int type, int pid,
   struct pid_namespace *ns);

-extern struct task_struct *find_task_by_pid(pid_t nr);
+extern struct task_struct *find_task_by_pid(pid_t nr) __deprecated;
 extern struct task_struct *find_task_by_vpid(pid_t nr);
 extern struct task_struct *find_task_by_pid_ns(pid_t nr,
   struct pid_namespace *ns);
diff --git a/init/main.c b/init/main.c
index e702632..38a16ba 100644
--- a/init/main.c
+++ b/init/main.c
@@ -437,7 +437,7 @@ static void noinline __init_refok rest_init(void)
  kernel_thread(kernel_init, NULL, CLONE_FS | CLONE_SIGHAND);
  numa_default_policy();
  pid = kernel_thread(kthreadd, NULL, CLONE_FS | CLONE_FILES);
- kthreadd_task = find_task_by_pid(pid);
+ kthreadd_task = find_task_by_pid_ns(pid, &init_pid_ns);
  unlock_kernel();

  /*
diff --git a/kernel/kgdb.c b/kernel/kgdb.c
index 87b0463..6de0fd0 100644
--- a/kernel/kgdb.c
+++ b/kernel/kgdb.c
@@ -621,7 +621,7 @@ static struct task_struct *getthread(struct pt_regs *regs, int tid)
  if (!tid)
   return NULL;

- return find_task_by_pid(tid);
+ return find_task_by_pid_ns(tid, &init_pid_ns);
 }

 #ifdef CONFIG_SMP
diff --git a/kernel/kthread.c b/kernel/kthread.c
index 0ac8878..b9caa46 100644
--- a/kernel/kthread.c
+++ b/kernel/kthread.c
@@ -99,7 +99,7 @@ static void create_kthread(struct kthread_create_info *create)
   struct sched_param param = { .sched_priority = 0 };
```

```
   wait_for_completion(&create->started);
   read_lock(&tasklist_lock);
-  create->result = find_task_by_pid(pid);
+  create->result = find_task_by_pid_ns(pid, &init_pid_ns);
   read_unlock(&tasklist_lock);
   /*
    * root may have changed our (kthreadd's) priority or CPU mask.
--
1.5.3.4
```

Subject: Re: [PATCH 6/12] gfs2: make gfs2_glock.gl_owner_pid be a struct pid *
Posted by Steven Whitehouse on Tue, 29 Jan 2008 14:19:45 GMT
View Forum Message <> Reply to Message

Hi,

Would you like me to take these two GFS2 patches into my git tree for
the next merge window, or are you intending that these should go to
Linus in this merge window? Either way I'm happy to add my:

Signed-off-by: Steven Whitehouse <swhiteho@redhat.com>

Steve.

On Tue, 2008-01-29 at 16:55 +0300, Pavel Emelyanov wrote:
> The gl_owner_pid field is used to get the lock owning
> task by its pid, so make it in a proper manner, i.e.
> by using the struct pid pointer and pid_task() function.
>
> The pid_task() becomes exported for the gfs2 module.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
> fs/gfs2/glock.c | 19 ++++++++++++-------
> fs/gfs2/incore.h | 2 +-
> 2 files changed, 13 insertions(+), 8 deletions(-)
>
> diff --git a/fs/gfs2/glock.c b/fs/gfs2/glock.c
> index 80e09c5..5fe585f 100644
> --- a/fs/gfs2/glock.c
> +++ b/fs/gfs2/glock.c
> @@ -334,7 +334,7 @@ int gfs2_glock_get(struct gfs2_sbd *sdp, u64 number,
>   gl->gl_state = LM_ST_UNLOCKED;
>   gl->gl_demote_state = LM_ST_EXCLUSIVE;
>   gl->gl_hash = hash;
> - gl->gl_owner_pid = 0;

```
> + gl->gl_owner_pid = NULL;
>   gl->gl_ip = 0;
>   gl->gl_ops = glops;
>   gl->gl_req_gh = NULL;
> @@ -631,7 +631,7 @@ static void gfs2_glmutex_lock(struct gfs2_glock *gl)
>    wait_on_holder(&gh);
>    gfs2_holder_uninit(&gh);
>   } else {
> -  gl->gl_owner_pid = current->pid;
> +  gl->gl_owner_pid = get_pid(task_pid(current));
>    gl->gl_ip = (unsigned long)__builtin_return_address(0);
>    spin_unlock(&gl->gl_spin);
>   }
> @@ -652,7 +652,7 @@ static int gfs2_glmutex_trylock(struct gfs2_glock *gl)
>   if (test_and_set_bit(GLF_LOCK, &gl->gl_flags)) {
>    acquired = 0;
>   } else {
> -  gl->gl_owner_pid = current->pid;
> +  gl->gl_owner_pid = get_pid(task_pid(current));
>    gl->gl_ip = (unsigned long)__builtin_return_address(0);
>   }
>   spin_unlock(&gl->gl_spin);
> @@ -668,12 +668,17 @@ static int gfs2_glmutex_trylock(struct gfs2_glock *gl)
>
>  static void gfs2_glmutex_unlock(struct gfs2_glock *gl)
>  {
> + struct pid *pid;
> +
>   spin_lock(&gl->gl_spin);
>   clear_bit(GLF_LOCK, &gl->gl_flags);
> - gl->gl_owner_pid = 0;
> + pid = gl->gl_owner_pid;
> + gl->gl_owner_pid = NULL;
>   gl->gl_ip = 0;
>   run_queue(gl);
>   spin_unlock(&gl->gl_spin);
> +
> + put_pid(pid);
>  }
>
>  /**
> @@ -1877,13 +1882,13 @@ static int dump_glock(struct glock_iter *gi, struct gfs2_glock *gl)
>   print_dbg(gi, "  gl_ref = %d\n", atomic_read(&gl->gl_ref));
>   print_dbg(gi, "  gl_state = %u\n", gl->gl_state);
>   if (gl->gl_owner_pid) {
> -  gl_owner = find_task_by_pid(gl->gl_owner_pid);
> +  gl_owner = pid_task(gl->gl_owner_pid, PIDTYPE_PID);
>    if (gl_owner)
```

```
>     print_dbg(gi, "  gl_owner = pid %d (%s)\n",
> -       gl->gl_owner_pid, gl_owner->comm);
> +       pid_nr(gl->gl_owner_pid), gl_owner->comm);
>     else
>       print_dbg(gi, "  gl_owner = %d (ended)\n",
> -       gl->gl_owner_pid);
> +       pid_nr(gl->gl_owner_pid));
>   } else
>     print_dbg(gi, "  gl_owner = -1\n");
>   print_dbg(gi, "  gl_ip = %lu\n", gl->gl_ip);
> diff --git a/fs/gfs2/incore.h b/fs/gfs2/incore.h
> index 1339996..8ad1c3f 100644
> --- a/fs/gfs2/incore.h
> +++ b/fs/gfs2/incore.h
> @@ -182,7 +182,7 @@ struct gfs2_glock {
>   unsigned int gl_hash;
>   unsigned int gl_demote_state; /* state requested by remote node */
>   unsigned long gl_demote_time; /* time of first demote request */
> - pid_t gl_owner_pid;
> + struct pid *gl_owner_pid;
>   unsigned long gl_ip;
>   struct list_head gl_holders;
>   struct list_head gl_waiters1; /* HIF_MUTEX */
> diff --git a/kernel/pid.c b/kernel/pid.c
> index e4e5fc8..4776915 100644
> --- a/kernel/pid.c
> +++ b/kernel/pid.c
> @@ -367,6 +367,7 @@ struct task_struct *pid_task(struct pid *pid, enum pid_type type)
>   }
>   return result;
>  }
> +EXPORT_SYMBOL(pid_task);
>
>  /*
>   * Must be called under rcu_read_lock() or with tasklist_lock read-held.
```

Subject: Re: [PATCH 5/12] Handle pid namespaces in cgroups code
Posted by Paul Menage on Tue, 29 Jan 2008 18:08:12 GMT
View Forum Message <> Reply to Message

On Jan 29, 2008 5:52 AM, Pavel Emelyanov <xemul@openvz.org> wrote:
> There's one place that works with task pids - its the "tasks" file
> in cgroups. The read/write handlers assume, that the pid values
> go to/come from the user space and thus it is a virtual pid, i.e.
> the pid as it is seen from inside a namespace.
>
> Tune the code accordingly.

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

Acked-by: Paul Menage <menage@google.com>

>
> ---
> kernel/cgroup.c | 4 ++--
> 1 files changed, 2 insertions(+), 2 deletions(-)
>
> diff --git a/kernel/cgroup.c b/kernel/cgroup.c
> index 2c5cccb..4766bb6 100644
> --- a/kernel/cgroup.c
> +++ b/kernel/cgroup.c
> @@ -1269,7 +1269,7 @@ static int attach_task_by_pid(struct cgroup *cgrp, char *pidbuf)
>
>   if (pid) {
>   rcu_read_lock();
> -   tsk = find_task_by_pid(pid);
> +   tsk = find_task_by_vpid(pid);
>   if (!tsk || tsk->flags & PF_EXITING) {
>   rcu_read_unlock();
>   return -ESRCH;
> @@ -1955,7 +1955,7 @@ static int pid_array_load(pid_t *pidarray, int npids, struct cgroup
*cgrp)
>   while ((tsk = cgroup_iter_next(cgrp, &it))) {
>   if (unlikely(n == npids))
>   break;
> -   pidarray[n++] = task_pid_nr(tsk);
> +   pidarray[n++] = task_pid_vnr(tsk);
>   }
>   cgroup_iter_end(cgrp, &it);
>   return n;
> --
> 1.5.3.4
>
>

Subject: Re: [PATCH 12/12] Deprecate the find_task_by_pid
Posted by akpm on Fri, 01 Feb 2008 22:27:50 GMT
View Forum Message <> Reply to Message

On Tue, 29 Jan 2008 17:09:55 +0300
Pavel Emelyanov <xemul@openvz.org> wrote:

> diff --git a/kernel/kgdb.c b/kernel/kgdb.c
> index 87b0463..6de0fd0 100644

> --- a/kernel/kgdb.c
> +++ b/kernel/kgdb.c
> @@ -621,7 +621,7 @@ static struct task_struct *getthread(struct pt_regs *regs, int tid)
> if (!tid)
> return NULL;
>
> - return find_task_by_pid(tid);
> + return find_task_by_pid_ns(tid, &init_pid_ns);
> }
>
> #ifdef CONFIG_SMP

The code I currently have here has changed. Is has grown a nice comment:

```
 /*
  * find_task_by_pid() does not take the tasklist lock anymore
  * but is nicely RCU locked - hence is a pretty resilient
  * thing to use:
  */
 return find_task_by_pid(tid);
```

I updated the comment appropriately
(s/find_task_by_pid/find_task_by_pid_ns/). Please check that this was
true.
