Subject: [PATCH 0/3] Parts of pid namespaces approved by all the
Posted by Pavel Emelianov on Tue, 10 Jul 2007 14:19:13 GMT
View Forum Message <> Reply to Message

These patches were acked by all the people who is interested in
having pid namespace in the kernel and proposed for inclusion.
Made against 2.6.22-rc6-mm1 and checked on i386 and x86_64 nodes.

[PATCH 1/3] - Api rename.
Acked by Cedric Le Goater.
[PATCH 2/3] - Small improvement of get_pid_ns() call.
Acked by Cedric.
[PATCH 3/3] - Kmem caches for infinite namespaces nesting.
Acked by Cedric and Sukadev Bhattiprolu.

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

Subject: [PATCH 1/3] Round up the API
Posted by Pavel Emelianov on Tue, 10 Jul 2007 14:21:38 GMT
View Forum Message <> Reply to Message

The set of functions process_session, task_session, process_group
and task_pgrp is confusing, as the names can be mixed with each other
when looking at the code for a long time.

The proposals are to
* equip the functions that return the integer with _nr suffix to
represent that fact,
* and to make all functions work with task (not process) by making
the common prefix of the same name.

For monotony the routines signal_session() and set_signal_session()
are replaced with task_session_nr() and set_task_session(), especially
since they are only used with the explicit task->signal dereference.

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

---

diff --git a/arch/mips/kernel/irixelf.c b/arch/mips/kernel/irixelf.c
index 403d96f..10ba0a5 100644
--- a/arch/mips/kernel/irixelf.c
+++ b/arch/mips/kernel/irixelf.c
@@ -1170,8 +1170,8 @@ static int irix_core_dump(long signr, st

```
  prstatus.pr_sighold = current->blocked.sig[0];
  psinfo.pr_pid = prstatus.pr_pid = current->pid;
  psinfo.pr_ppid = prstatus.pr_ppid = current->parent->pid;
- psinfo.pr_pgrp = prstatus.pr_pgrp = process_group(current);
- psinfo.pr_sid = prstatus.pr_sid = process_session(current);
+ psinfo.pr_pgrp = prstatus.pr_pgrp = task_pgrp_nr(current);
+ psinfo.pr_sid = prstatus.pr_sid = task_session_nr(current);
  if (current->pid == current->tgid) {
   /*
    * This is the record for the group leader.  Add in the
diff --git a/arch/mips/kernel/irixsig.c b/arch/mips/kernel/irixsig.c
index 6980deb..210503e 100644
--- a/arch/mips/kernel/irixsig.c
+++ b/arch/mips/kernel/irixsig.c
@@ -609,7 +609,7 @@ repeat:
   p = list_entry(_p,struct task_struct,sibling);
   if ((type == IRIX_P_PID) && p->pid != pid)
    continue;
-  if ((type == IRIX_P_PGID) && process_group(p) != pid)
+  if ((type == IRIX_P_PGID) && task_pgrp_nr(p) != pid)
    continue;
   if ((p->exit_signal != SIGCHLD))
    continue;
diff --git a/arch/mips/kernel/sysirix.c b/arch/mips/kernel/sysirix.c
index 93a1484..23c3e82 100644
--- a/arch/mips/kernel/sysirix.c
+++ b/arch/mips/kernel/sysirix.c
@@ -763,11 +763,11 @@ asmlinkage int irix_setpgrp(int flags)
  printk("[%s:%d] setpgrp(%d) ", current->comm, current->pid, flags);
 #endif
  if(!flags)
-  error = process_group(current);
+  error = task_pgrp_nr(current);
  else
   error = sys_setsid();
 #ifdef DEBUG_PROCGRPS
- printk("returning %d\n", process_group(current));
+ printk("returning %d\n", task_pgrp_nr(current));
 #endif

  return error;
diff --git a/arch/sparc64/solaris/misc.c b/arch/sparc64/solaris/misc.c
index 3b67de7..c86cb30 100644
--- a/arch/sparc64/solaris/misc.c
+++ b/arch/sparc64/solaris/misc.c
@@ -415,7 +415,7 @@ asmlinkage int solaris_procids(int cmd,

  switch (cmd) {
```

```
  case 0: /* getpgrp */
-  return process_group(current);
+  return task_pgrp_nr(current);
  case 1: /* setpgrp */
   {
    int (*sys_setpgid)(pid_t,pid_t) =
@@ -426,7 +426,7 @@ asmlinkage int solaris_procids(int cmd,
    ret = sys_setpgid(0, 0);
    if (ret) return ret;
    proc_clear_tty(current);
-   return process_group(current);
+   return task_pgrp_nr(current);
   }
  case 2: /* getsid */
   {
diff --git a/drivers/char/tty_io.c b/drivers/char/tty_io.c
index de37ebc..a58ea84 100644
--- a/drivers/char/tty_io.c
+++ b/drivers/char/tty_io.c
@@ -3501,7 +3501,7 @@ void __do_SAK(struct tty_struct *tty)
  /* Kill the entire session */
  do_each_pid_task(session, PIDTYPE_SID, p) {
   printk(KERN_NOTICE "SAK: killed process %d"
-   " (%s): process_session(p)==tty->session\n",
+   " (%s): task_session_nr(p)==tty->session\n",
    p->pid, p->comm);
   send_sig(SIGKILL, p, 1);
  } while_each_pid_task(session, PIDTYPE_SID, p);
@@ -3511,7 +3511,7 @@ void __do_SAK(struct tty_struct *tty)
  do_each_thread(g, p) {
   if (p->signal->tty == tty) {
    printk(KERN_NOTICE "SAK: killed process %d"
-       " (%s): process_session(p)==tty->session\n",
+       " (%s): task_session_nr(p)==tty->session\n",
        p->pid, p->comm);
    send_sig(SIGKILL, p, 1);
    continue;
diff --git a/fs/autofs/inode.c b/fs/autofs/inode.c
index e7204d7..45f5992 100644
--- a/fs/autofs/inode.c
+++ b/fs/autofs/inode.c
@@ -80,7 +80,7 @@ static int parse_options(char *options,

  *uid = current->uid;
  *gid = current->gid;
- *pgrp = process_group(current);
+ *pgrp = task_pgrp_nr(current);
```

```
  *minproto = *maxproto = AUTOFS_PROTO_VERSION;

diff --git a/fs/autofs/root.c b/fs/autofs/root.c
index c148953..592f640 100644
--- a/fs/autofs/root.c
+++ b/fs/autofs/root.c
@@ -215,7 +215,7 @@ static struct dentry *autofs_root_lookup
  oz_mode = autofs_oz_mode(sbi);
  DPRINTK(("autofs_lookup: pid = %u, pgrp = %u, catatonic = %d, "
     "oz_mode = %d\n", pid_nr(task_pid(current)),
-    process_group(current), sbi->catatonic,
+    task_pgrp_nr(current), sbi->catatonic,
     oz_mode));

  /*
@@ -536,7 +536,7 @@ static int autofs_root_ioctl(struct inod
  struct autofs_sb_info *sbi = autofs_sbi(inode->i_sb);
  void __user *argp = (void __user *)arg;

- DPRINTK(("autofs_ioctl: cmd = 0x%08x, arg = 0x%08lx, sbi = %p, pgrp =
%u\n",cmd,arg,sbi,process_group(current)));
+ DPRINTK(("autofs_ioctl: cmd = 0x%08x, arg = 0x%08lx, sbi = %p, pgrp =
%u\n",cmd,arg,sbi,task_pgrp_nr(current)));

  if (_IOC_TYPE(cmd) != _IOC_TYPE(AUTOFS_IOC_FIRST) ||
      _IOC_NR(cmd) - _IOC_NR(AUTOFS_IOC_FIRST) >= AUTOFS_IOC_COUNT)
diff --git a/fs/autofs4/autofs_i.h b/fs/autofs4/autofs_i.h
index d85f42f..2d4ae40 100644
--- a/fs/autofs4/autofs_i.h
+++ b/fs/autofs4/autofs_i.h
@@ -131,7 +131,7 @@ static inline struct autofs_info *autofs
    filesystem without "magic".) */

 static inline int autofs4_oz_mode(struct autofs_sb_info *sbi) {
- return sbi->catatonic || process_group(current) == sbi->oz_pgrp;
+ return sbi->catatonic || task_pgrp_nr(current) == sbi->oz_pgrp;
 }

 /* Does a dentry have some pending activity? */
diff --git a/fs/autofs4/inode.c b/fs/autofs4/inode.c
index 692364e..32a39b0 100644
--- a/fs/autofs4/inode.c
+++ b/fs/autofs4/inode.c
@@ -226,7 +226,7 @@ static int parse_options(char *options,

  *uid = current->uid;
  *gid = current->gid;
- *pgrp = process_group(current);
```

```
+ *pgrp = task_pgrp_nr(current);

  *minproto = AUTOFS_MIN_PROTO_VERSION;
  *maxproto = AUTOFS_MAX_PROTO_VERSION;
@@ -325,7 +325,7 @@ int autofs4_fill_super(struct super_bloc
  sbi->pipe = NULL;
  sbi->catatonic = 1;
  sbi->exp_timeout = 0;
- sbi->oz_pgrp = process_group(current);
+ sbi->oz_pgrp = task_pgrp_nr(current);
  sbi->sb = s;
  sbi->version = 0;
  sbi->sub_version = 0;
diff --git a/fs/autofs4/root.c b/fs/autofs4/root.c
index 2d4c8a3..c766ff8 100644
--- a/fs/autofs4/root.c
+++ b/fs/autofs4/root.c
@@ -582,7 +582,7 @@ static struct dentry *autofs4_lookup(str
  oz_mode = autofs4_oz_mode(sbi);

  DPRINTK("pid = %u, pgrp = %u, catatonic = %d, oz_mode = %d",
-   current->pid, process_group(current), sbi->catatonic, oz_mode);
+   current->pid, task_pgrp_nr(current), sbi->catatonic, oz_mode);

  unhashed = autofs4_lookup_unhashed(sbi, dentry->d_parent, &dentry->d_name);
  if (!unhashed) {
@@ -973,7 +973,7 @@ static int autofs4_root_ioctl(struct ino
  void __user *p = (void __user *)arg;

  DPRINTK("cmd = 0x%08x, arg = 0x%08lx, sbi = %p, pgrp = %u",
-  cmd,arg,sbi,process_group(current));
+  cmd,arg,sbi,task_pgrp_nr(current));

  if (_IOC_TYPE(cmd) != _IOC_TYPE(AUTOFS_IOC_FIRST) ||
      _IOC_NR(cmd) - _IOC_NR(AUTOFS_IOC_FIRST) >= AUTOFS_IOC_COUNT)
diff --git a/fs/binfmt_elf.c b/fs/binfmt_elf.c
index da270d1..0777ee1 100644
--- a/fs/binfmt_elf.c
+++ b/fs/binfmt_elf.c
@@ -1404,8 +1404,8 @@ static void fill_prstatus(struct elf_prs
  prstatus->pr_sighold = p->blocked.sig[0];
  prstatus->pr_pid = p->pid;
  prstatus->pr_ppid = p->parent->pid;
- prstatus->pr_pgrp = process_group(p);
- prstatus->pr_sid = process_session(p);
+ prstatus->pr_pgrp = task_pgrp_nr(p);
+ prstatus->pr_sid = task_session_nr(p);
  if (thread_group_leader(p)) {
```

```
   /*
    * This is the record for the group leader.  Add in the
@@ -1450,8 +1450,8 @@ static int fill_psinfo(struct elf_prpsin

  psinfo->pr_pid = p->pid;
  psinfo->pr_ppid = p->parent->pid;
- psinfo->pr_pgrp = process_group(p);
- psinfo->pr_sid = process_session(p);
+ psinfo->pr_pgrp = task_pgrp_nr(p);
+ psinfo->pr_sid = task_session_nr(p);

  i = p->state ? ffz(~p->state) + 1 : 0;
  psinfo->pr_state = i;
diff --git a/fs/binfmt_elf_fdpic.c b/fs/binfmt_elf_fdpic.c
index 2f5d8db..46c1d03 100644
--- a/fs/binfmt_elf_fdpic.c
+++ b/fs/binfmt_elf_fdpic.c
@@ -1344,8 +1344,8 @@ static void fill_prstatus(struct elf_prs
  prstatus->pr_sighold = p->blocked.sig[0];
  prstatus->pr_pid = p->pid;
  prstatus->pr_ppid = p->parent->pid;
- prstatus->pr_pgrp = process_group(p);
- prstatus->pr_sid = process_session(p);
+ prstatus->pr_pgrp = task_pgrp_nr(p);
+ prstatus->pr_sid = task_session_nr(p);
  if (thread_group_leader(p)) {
   /*
    * This is the record for the group leader.  Add in the
@@ -1393,8 +1393,8 @@ static int fill_psinfo(struct elf_prpsin

  psinfo->pr_pid = p->pid;
  psinfo->pr_ppid = p->parent->pid;
- psinfo->pr_pgrp = process_group(p);
- psinfo->pr_sid = process_session(p);
+ psinfo->pr_pgrp = task_pgrp_nr(p);
+ psinfo->pr_sid = task_session_nr(p);

  i = p->state ? ffz(~p->state) + 1 : 0;
  psinfo->pr_state = i;
diff --git a/fs/coda/upcall.c b/fs/coda/upcall.c
index 5faacdb..89b3073 100644
--- a/fs/coda/upcall.c
+++ b/fs/coda/upcall.c
@@ -53,7 +53,7 @@ static void *alloc_upcall(int opcode, in

        inp->ih.opcode = opcode;
  inp->ih.pid = current->pid;
- inp->ih.pgid = process_group(current);
```

```
+ inp->ih.pgid = task_pgrp_nr(current);
 #ifdef CONFIG_CODA_FS_OLD_API
  memset(&inp->ih.cred, 0, sizeof(struct coda_cred));
  inp->ih.cred.cr_fsuid = current->fsuid;
diff --git a/fs/proc/array.c b/fs/proc/array.c
index b4c8965..d9c0766 100644
--- a/fs/proc/array.c
+++ b/fs/proc/array.c
@@ -428,8 +428,8 @@ static int do_task_stat(struct task_stru
     stime += cputime_to_clock_t(sig->stime);
   }

-  sid = signal_session(sig);
-  pgid = process_group(task);
+  sid = task_session_nr(task);
+  pgid = task_pgrp_nr(task);
   ppid = rcu_dereference(task->real_parent)->tgid;

   unlock_task_sighand(task, &flags);
diff --git a/include/linux/sched.h b/include/linux/sched.h
index e953c55..69e9b3a 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1170,24 +1170,19 @@ struct task_struct {
 #endif
 };

-static inline pid_t process_group(struct task_struct *tsk)
+static inline pid_t task_pgrp_nr(struct task_struct *tsk)
 {
  return tsk->signal->pgrp;
 }

-static inline pid_t signal_session(struct signal_struct *sig)
+static inline pid_t task_session_nr(struct task_struct *tsk)
 {
- return sig->__session;
+ return tsk->signal->__session;
 }

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

```
- sig->__session = session;
+ tsk->signal->__session = session;
 }

 static inline struct pid *task_pid(struct task_struct *task)
diff --git a/kernel/exit.c b/kernel/exit.c
index af031a7..4dc4122 100644
--- a/kernel/exit.c
+++ b/kernel/exit.c
@@ -309,12 +309,12 @@ void __set_special_pids(pid_t session, p
 {
  struct task_struct *curr = current->group_leader;

- if (process_session(curr) != session) {
+ if (task_session_nr(curr) != session) {
   detach_pid(curr, PIDTYPE_SID);
-  set_signal_session(curr->signal, session);
+  set_task_session(curr, session);
   attach_pid(curr, PIDTYPE_SID, find_pid(session));
  }
- if (process_group(curr) != pgrp) {
+ if (task_pgrp_nr(curr) != pgrp) {
   detach_pid(curr, PIDTYPE_PGID);
   curr->signal->pgrp = pgrp;
   attach_pid(curr, PIDTYPE_PGID, find_pid(pgrp));
@@ -1108,10 +1108,10 @@ static int eligible_child(pid_t pid, int
   if (p->pid != pid)
    return 0;
  } else if (!pid) {
-  if (process_group(p) != process_group(current))
+  if (task_pgrp_nr(p) != task_pgrp_nr(current))
    return 0;
  } else if (pid != -1) {
-  if (process_group(p) != -pid)
+  if (task_pgrp_nr(p) != -pid)
    return 0;
  }

diff --git a/kernel/fork.c b/kernel/fork.c
index 43744ba..e7e2d0c 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -1262,8 +1262,8 @@ static struct task_struct *copy_process(

   if (thread_group_leader(p)) {
    p->signal->tty = current->signal->tty;
-   p->signal->pgrp = process_group(current);
-   set_signal_session(p->signal, process_session(current));
```

```
+   p->signal->pgrp = task_pgrp_nr(current);
+   set_task_session(p, task_session_nr(current));
    attach_pid(p, PIDTYPE_PGID, task_pgrp(current));
    attach_pid(p, PIDTYPE_SID, task_session(current));

diff --git a/kernel/signal.c b/kernel/signal.c
index ef8156a..dae013d 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -540,7 +540,7 @@ static int check_kill_permission(int sig
  error = -EPERM;
  if ((info == SEND_SIG_NOINFO || (!is_si_special(info) && SI_FROMUSER(info)))
      && ((sig != SIGCONT) ||
-  (process_session(current) != process_session(t)))
+  (task_session_nr(current) != task_session_nr(t)))
      && (current->euid ^ t->suid) && (current->euid ^ t->uid)
      && (current->uid ^ t->suid) && (current->uid ^ t->uid)
      && !capable(CAP_KILL))
diff --git a/kernel/sys.c b/kernel/sys.c
index 01d6dac..c5ef39d 100644
--- a/kernel/sys.c
+++ b/kernel/sys.c
@@ -1487,7 +1487,7 @@ asmlinkage long sys_setpgid(pid_t pid, p
  if (err)
   goto out;

- if (process_group(p) != pgid) {
+ if (task_pgrp_nr(p) != pgid) {
   detach_pid(p, PIDTYPE_PGID);
   p->signal->pgrp = pgid;
   attach_pid(p, PIDTYPE_PGID, find_pid(pgid));
@@ -1503,7 +1503,7 @@ out:
 asmlinkage long sys_getpgid(pid_t pid)
 {
  if (!pid)
-  return process_group(current);
+  return task_pgrp_nr(current);
  else {
   int retval;
   struct task_struct *p;
@@ -1515,7 +1515,7 @@ asmlinkage long sys_getpgid(pid_t pid)
   if (p) {
    retval = security_task_getpgid(p);
    if (!retval)
-    retval = process_group(p);
+    retval = task_pgrp_nr(p);
   }
   read_unlock(&tasklist_lock);
```

```
   return retval;
@@ -1527,7 +1527,7 @@ asmlinkage long sys_getpgid(pid_t pid)
 asmlinkage long sys_getpgrp(void)
 {
  /* SMP - assuming writes are word atomic this is fine */
- return process_group(current);
+ return task_pgrp_nr(current);
 }

 #endif
@@ -1535,7 +1535,7 @@ asmlinkage long sys_getpgrp(void)
 asmlinkage long sys_getsid(pid_t pid)
 {
  if (!pid)
-  return process_session(current);
+  return task_session_nr(current);
  else {
   int retval;
   struct task_struct *p;
@@ -1547,7 +1547,7 @@ asmlinkage long sys_getsid(pid_t pid)
   if (p) {
    retval = security_task_getsid(p);
    if (!retval)
-    retval = process_session(p);
+    retval = task_session_nr(p);
   }
   read_unlock(&tasklist_lock);
   return retval;
@@ -1584,7 +1584,7 @@ asmlinkage long sys_setsid(void)
  group_leader->signal->tty = NULL;
  spin_unlock(&group_leader->sighand->siglock);

- err = process_group(group_leader);
+ err = task_pgrp_nr(group_leader);
 out:
  write_unlock_irq(&tasklist_lock);
  return err;
```

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

---

Subject: [PATCH 2/3] Make get_pid_ns() return the namespace itself
Posted by Pavel Emelianov on Tue, 10 Jul 2007 14:23:29 GMT
View Forum Message <> Reply to Message

Make get_pid_ns() return the namespace itself to look like

the other getters and make the code using it look nicer.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>
Acked-by: Cedric Le Goater <clg@fr.ibm.com>

```
diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
index b9a17e0..ddb9a4c 100644
--- a/include/linux/pid_namespace.h
+++ b/include/linux/pid_namespace.h
@@ -24,9 +24,10 @@ struct pid_namespace {

 extern struct pid_namespace init_pid_ns;

-static inline void get_pid_ns(struct pid_namespace *ns)
+static inline struct pid_namespace *get_pid_ns(struct pid_namespace *ns)
 {
  kref_get(&ns->kref);
+ return ns;
 }

 extern struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *ns);
```

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

Subject: [PATCH 3/3] Dynamic kmem cache allocator for pid namespaces
Posted by Pavel Emelianov on Tue, 10 Jul 2007 14:28:10 GMT
View Forum Message <> Reply to Message

Add kmem_cache to pid_namespace to allocate pids from.

Since booth implementations expand the struct pid to carry
more numerical values each namespace should have separate
cache to store pids of different sizes.

Each kmem cache is names "pid_<NR>", where <NR> is the number
of numerical ids on the pid. Different namespaces with same
level of nesting will have same caches.

This patch has two FIXMEs that are to be fixed after we reach
the consensus about the struct pid itself.

The first one is that the namespace to free the pid from in
free_pid() must be taken from pid. Now the init_pid_ns is

used.

The second FIXME is about the cache allocation. When we do know
how long the object will be then we'll have to calculate this
size in create_pid_cachep. Right now the sizeof(struct pid)
value is used.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>
Acked-by: Cedric Le Goater <clg@fr.ibm.com>
Acked-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>

```
diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
index ddb9a4c..27cfad3 100644
--- a/include/linux/pid_namespace.h
+++ b/include/linux/pid_namespace.h
@@ -20,6 +20,7 @@ struct pid_namespace {
  struct pidmap pidmap[PIDMAP_ENTRIES];
  int last_pid;
  struct task_struct *child_reaper;
+ struct kmem_cache_t *pid_cachep;
 };

 extern struct pid_namespace init_pid_ns;
diff --git a/kernel/pid.c b/kernel/pid.c
index c6e3f9f..d3f0f91 100644
--- a/kernel/pid.c
+++ b/kernel/pid.c
@@ -32,7 +32,6 @@
 #define pid_hashfn(nr) hash_long((unsigned long)nr, pidhash_shift)
 static struct hlist_head *pid_hash;
 static int pidhash_shift;
-static struct kmem_cache *pid_cachep;
 struct pid init_struct_pid = INIT_STRUCT_PID;

 int pid_max = PID_MAX_DEFAULT;
@@ -176,11 +175,16 @@ static int next_pidmap(struct pid_namesp

 fastcall void put_pid(struct pid *pid)
 {
+ struct pid_namespace *ns;
+
  if (!pid)
   return;
+
+ /* FIXME - this must be the namespace this pid lives in */
+ ns = &init_pid_ns;
```

```
  if ((atomic_read(&pid->count) == 1) ||
       atomic_dec_and_test(&pid->count))
-  kmem_cache_free(pid_cachep, pid);
+  kmem_cache_free(ns->pid_cachep, pid);
 }
 EXPORT_SYMBOL_GPL(put_pid);

@@ -208,12 +212,14 @@ struct pid *alloc_pid(void)
  struct pid *pid;
  enum pid_type type;
  int nr = -1;
+ struct pid_namespace *ns;

- pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
+ ns = current->nsproxy->pid_ns;
+ pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
  if (!pid)
   goto out;

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

@@ -230,7 +236,7 @@ out:
  return pid;

 out_free:
- kmem_cache_free(pid_cachep, pid);
+ kmem_cache_free(ns->pid_cachep, pid);
  pid = NULL;
  goto out;
 }
@@ -365,6 +371,56 @@ struct pid *find_ge_pid(int nr)
 }
 EXPORT_SYMBOL_GPL(find_get_pid);

+struct pid_cache {
+ int nr_ids;
+ char name[16];
+ struct kmem_cache *cachep;
+ struct list_head list;
+};
+
+static LIST_HEAD(pid_caches_lh);
+static DEFINE_MUTEX(pid_caches_mutex);
+
+/*
```

```
+ * creates the kmem cache to allocate pids from.
+ * @nr_ids: the number of numerical ids this pid will have to carry
+ */
+
+static struct kmem_cache *create_pid_cachep(int nr_ids)
+{
+ struct pid_cache *pcache;
+ struct kmem_cache *cachep;
+
+ mutex_lock(&pid_caches_mutex);
+ list_for_each_entry (pcache, &pid_caches_lh, list)
+  if (pcache->nr_ids == nr_ids)
+   goto out;
+
+ pcache = kmalloc(sizeof(struct pid_cache), GFP_KERNEL);
+ if (pcache == NULL)
+  goto err_alloc;
+
+ snprintf(pcache->name, sizeof(pcache->name), "pid_%d", nr_ids);
+ cachep = kmem_cache_create(pcache->name,
+   /* FIXME add numerical ids here */
+   sizeof(struct pid), 0, SLAB_HWCACHE_ALIGN, NULL, NULL);
+ if (cachep == NULL)
+  goto err_cachep;
+
+ pcache->nr_ids = nr_ids;
+ pcache->cachep = cachep;
+ list_add(&pcache->list, &pid_caches_lh);
+out:
+ mutex_unlock(&pid_caches_mutex);
+ return pcache->cachep;
+
+err_cachep:
+ kfree(pcache);
+err_alloc:
+ mutex_unlock(&pid_caches_mutex);
+ return NULL;
+}
+
 struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *old_ns)
 {
  BUG_ON(!old_ns);
@@ -412,5 +468,7 @@ void __init pidmap_init(void)
  set_bit(0, init_pid_ns.pidmap[0].page);
  atomic_dec(&init_pid_ns.pidmap[0].nr_free);

- pid_cachep = KMEM_CACHE(pid, SLAB_PANIC);
+ init_pid_ns.pid_cachep = create_pid_cachep(1);
```

```
+ if (init_pid_ns.pid_cachep == NULL)
+  panic("Can't create pid_1 cachep\n");
 }
```

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

---

Subject: Re: [PATCH 0/3] Parts of pid namespaces approved by all the
Posted by Pavel Emelianov on Tue, 10 Jul 2007 14:30:05 GMT
View Forum Message <> Reply to Message

My apologizes, but I have just noticed that the subject of the
very first letter was accidentally cut (by my mailer?).

The correct one is

"[PATCH 0/3] Parts of pid namespaces approved by all the participants"

Thanks,
Pavel

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

---

Subject: Re: [PATCH 3/3] Dynamic kmem cache allocator for pid namespaces
Posted by akpm on Thu, 12 Jul 2007 22:47:50 GMT
View Forum Message <> Reply to Message

On Tue, 10 Jul 2007 18:28:10 +0400
Pavel Emelianov <xemul@openvz.org> wrote:

> Add kmem_cache to pid_namespace to allocate pids from.

I'm having trouble understanding this changelog.

> Since booth implementations expand the struct pid to carry
> more numerical values each namespace should have separate
> cache to store pids of different sizes.

Assuming "booth" means "both": you are referring to two different
implementations of <something>. What are they? Where are they?

I don't understand any of this :(

> Each kmem cache is names "pid_<NR>", where <NR> is the number
> of numerical ids on the pid.

What is a "numerical ID on a pid"?

> Different namespaces with same
> level of nesting will have same caches.

ok...

> This patch has two FIXMEs that are to be fixed after we reach
> the consensus about the struct pid itself.
>
> The first one is that the namespace to free the pid from in
> free_pid() must be taken from pid. Now the init_pid_ns is
> used.

That looks like a fatal bug to me? We free a slab object into a kmem_cache
which it was not obtained from? slab will go BUG, surely?

> The second FIXME is about the cache allocation. When we do know
> how long the object will be then we'll have to calculate this
> size in create_pid_cachep. Right now the sizeof(struct pid)
> value is used.

hm, we already have code which is good at choosing an appropriate cache
based upon the requested size. It's called kmalloc() ;)

Do we really expect that there will be so many of these objects that the
(modest) space-saving which a custom cache provides will be worthwhile?

> diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
> index ddb9a4c..27cfad3 100644
> --- a/include/linux/pid_namespace.h
> +++ b/include/linux/pid_namespace.h
> @@ -20,6 +20,7 @@ struct pid_namespace {
>   struct pidmap pidmap[PIDMAP_ENTRIES];
>   int last_pid;
>   struct task_struct *child_reaper;
> + struct kmem_cache_t *pid_cachep;
> };
>
> extern struct pid_namespace init_pid_ns;
> diff --git a/kernel/pid.c b/kernel/pid.c
> index c6e3f9f..d3f0f91 100644
> --- a/kernel/pid.c
> +++ b/kernel/pid.c

```
> @@ -32,7 +32,6 @@
>  #define pid_hashfn(nr) hash_long((unsigned long)nr, pidhash_shift)
>  static struct hlist_head *pid_hash;
>  static int pidhash_shift;
> -static struct kmem_cache *pid_cachep;
>  struct pid init_struct_pid = INIT_STRUCT_PID;
>
>  int pid_max = PID_MAX_DEFAULT;
> @@ -176,11 +175,16 @@ static int next_pidmap(struct pid_namesp
>
>  fastcall void put_pid(struct pid *pid)
>  {
> + struct pid_namespace *ns;
> +
>   if (!pid)
>    return;
> +
> + /* FIXME - this must be the namespace this pid lives in */
> + ns = &init_pid_ns;
>   if ((atomic_read(&pid->count) == 1) ||
>       atomic_dec_and_test(&pid->count))
> -  kmem_cache_free(pid_cachep, pid);
> +  kmem_cache_free(ns->pid_cachep, pid);
>  }
>  EXPORT_SYMBOL_GPL(put_pid);
>
> @@ -208,12 +212,14 @@ struct pid *alloc_pid(void)
>   struct pid *pid;
>   enum pid_type type;
>   int nr = -1;
> + struct pid_namespace *ns;
>
> - pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
> + ns = current->nsproxy->pid_ns;
> + pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
>   if (!pid)
>    goto out;
>
> - nr = alloc_pidmap(current->nsproxy->pid_ns);
> + nr = alloc_pidmap(ns);
>   if (nr < 0)
>    goto out_free;
>
> @@ -230,7 +236,7 @@ out:
>   return pid;
>
>  out_free:
> - kmem_cache_free(pid_cachep, pid);
```

> + kmem_cache_free(ns->pid_cachep, pid);
>   pid = NULL;
>   goto out;
> }
> @@ -365,6 +371,56 @@ struct pid *find_ge_pid(int nr)
> }
> EXPORT_SYMBOL_GPL(find_get_pid);
>
> +struct pid_cache {
> + int nr_ids;
> + char name[16];
> + struct kmem_cache *cachep;
> + struct list_head list;
> +};

Please put a lot of effort into documenting data structures. They are key
to understanding the code and effort here really does pay off in
understandability and hence maintainability and quality.

I would contend that zero documentation simply does not meet kernel standards.

A good code comment would, amongst other things, explain to the reader why
this code exists at all. That would (permanently) answer my kmalloc
question, above.

> +static LIST_HEAD(pid_caches_lh);
> +static DEFINE_MUTEX(pid_caches_mutex);
> +
> +/*
> + * creates the kmem cache to allocate pids from.
> + * @nr_ids: the number of numerical ids this pid will have to carry
> + */
> +
> +static struct kmem_cache *create_pid_cachep(int nr_ids)
> +{
> + struct pid_cache *pcache;
> + struct kmem_cache *cachep;
> +
> + mutex_lock(&pid_caches_mutex);
> + list_for_each_entry (pcache, &pid_caches_lh, list)
> + if (pcache->nr_ids == nr_ids)
> +   goto out;
> +
> + pcache = kmalloc(sizeof(struct pid_cache), GFP_KERNEL);
> + if (pcache == NULL)
> + goto err_alloc;
> +
> + snprintf(pcache->name, sizeof(pcache->name), "pid_%d", nr_ids);

```
> + cachep = kmem_cache_create(pcache->name,
> +   /* FIXME add numerical ids here */
> +   sizeof(struct pid), 0, SLAB_HWCACHE_ALIGN, NULL, NULL);
> + if (cachep == NULL)
> +  goto err_cachep;
> +
> + pcache->nr_ids = nr_ids;
> + pcache->cachep = cachep;
> + list_add(&pcache->list, &pid_caches_lh);
> +out:
> + mutex_unlock(&pid_caches_mutex);
> + return pcache->cachep;
> +
> +err_cachep:
> + kfree(pcache);
> +err_alloc:
> + mutex_unlock(&pid_caches_mutex);
> + return NULL;
> +}
> +
>  struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *old_ns)
>  {
>   BUG_ON(!old_ns);
> @@ -412,5 +468,7 @@ void __init pidmap_init(void)
>   set_bit(0, init_pid_ns.pidmap[0].page);
>   atomic_dec(&init_pid_ns.pidmap[0].nr_free);
>
> - pid_cachep = KMEM_CACHE(pid, SLAB_PANIC);
> + init_pid_ns.pid_cachep = create_pid_cachep(1);
> + if (init_pid_ns.pid_cachep == NULL)
> +  panic("Can't create pid_1 cachep\n");
>  }
```

hm, so that global list of pid_caches which we're carefully maintaining
doesn't actually get used for anything.

I assume there is some plan to use this list in the future, but this should
have been covered in the changelog, please.

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

---

Subject: Re: [PATCH 3/3] Dynamic kmem cache allocator for pid namespaces
Posted by Sukadev Bhattiprolu on Sat, 14 Jul 2007 01:22:25 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@openvz.org] wrote:
| Add kmem_cache to pid_namespace to allocate pids from.
|
| Since booth implementations expand the struct pid to carry
| more numerical values each namespace should have separate
| cache to store pids of different sizes.
|
| Each kmem cache is names "pid_<NR>", where <NR> is the number
| of numerical ids on the pid. Different namespaces with same
| level of nesting will have same caches.
|
| This patch has two FIXMEs that are to be fixed after we reach
| the consensus about the struct pid itself.
|
| The first one is that the namespace to free the pid from in
| free_pid() must be taken from pid. Now the init_pid_ns is
| used.
|
| The second FIXME is about the cache allocation. When we do know
| how long the object will be then we'll have to calculate this
| size in create_pid_cachep. Right now the sizeof(struct pid)
| value is used.
|
| Signed-off-by: Pavel Emelianov <xemul@openvz.org>
| Acked-by: Cedric Le Goater <clg@fr.ibm.com>
| Acked-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
|
| ---
|
| diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
| index ddb9a4c..27cfad3 100644
| --- a/include/linux/pid_namespace.h
| +++ b/include/linux/pid_namespace.h
| @@ -20,6 +20,7 @@ struct pid_namespace {
| struct pidmap pidmap[PIDMAP_ENTRIES];
| int last_pid;
| struct task_struct *child_reaper;
| + struct kmem_cache_t *pid_cachep;

Shouldn't this be 'struct kmem_cache *' ?

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