
Subject: [PATCH 5/6] Cleanup the code managed with PID_NS option
Posted by [Pavel Emelianov](#) on Wed, 14 Nov 2007 11:39:00 GMT
[View Forum Message](#) <> [Reply to Message](#)

Just like with the user namespaces, move the namespace management code into the separate .c file and mark the (already existing) PID_NS option as "depend on NAMESPACES"

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

```
diff --git a/include/linux/pid.h b/include/linux/pid.h
index e29a900..061abb6 100644
--- a/include/linux/pid.h
+++ b/include/linux/pid.h
@@ -118,10 +118,10 @@ extern struct pid *find_pid(int nr);
 */
extern struct pid *find_get_pid(int nr);
extern struct pid *find_get_pid(int nr, struct pid_namespace *);
+int next_pidmap(struct pid_namespace *pid_ns, int last);

extern struct pid *alloc_pid(struct pid_namespace *ns);
extern void FASTCALL(free_pid(struct pid *pid));
-extern void zap_pid_ns_processes(struct pid_namespace *pid_ns);

/*
 * the helpers to get the pid's id seen from different namespaces
diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
index 1689e28..fcd61fa 100644
--- a/include/linux/pid_namespace.h
+++ b/include/linux/pid_namespace.h
@@ -39,6 +39,7 @@ @@ static inline struct pid_namespace *get_pid_ns(struct pid_namespace *ns)

extern struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *ns);
extern void free_pid_ns(struct kref *kref);
+extern void zap_pid_ns_processes(struct pid_namespace *pid_ns);

static inline void put_pid_ns(struct pid_namespace *ns)
{
@@ -66,6 +67,11 @@ @@ static inline void put_pid_ns(struct pid_namespace *ns)
{
}

+
+static inline void zap_pid_ns_processes(struct pid_namespace *ns)
+{
+ BUG();
```

```

+}
#endif /* CONFIG_PID_NS */

static inline struct pid_namespace *task_active_pid_ns(struct task_struct *tsk)
diff --git a/init/Kconfig b/init/Kconfig
index 825f10c..f21bc4d 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -206,18 +206,6 @@ config TASK_IO_ACCOUNTING

```

Say N if unsure.

```

-config PID_NS
- bool "PID Namespaces (EXPERIMENTAL)"
- default n
- depends on EXPERIMENTAL
- help
-   Support process id namespaces. This allows having multiple
-   process with the same pid as long as they are in different
-   pid namespaces. This is a building block of containers.
-
- Unless you want to work with an experimental feature
- say N here.
-
config AUDIT
 bool "Auditing support"
 depends on NET
@@ -426,6 +414,18 @@ config USER_NS
 to provide different user info for different servers.
 If unsure, say N.

```

```

+config PID_NS
+ bool "PID Namespaces (EXPERIMENTAL)"
+ default n
+ depends on NAMESPACES && EXPERIMENTAL
+ help
+   Support process id namespaces. This allows having multiple
+   process with the same pid as long as they are in different
+   pid namespaces. This is a building block of containers.
+
+ Unless you want to work with an experimental feature
+ say N here.
+
config BLK_DEV_INITRD
 bool "Initial RAM filesystem and RAM disk (initramfs/initrd) support"
 depends on BROKEN || !FRV
diff --git a/kernel/Makefile b/kernel/Makefile
index d01cb7b..d108027 100644

```

```

--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ -50,6 +50,7 @@ obj-$(CONFIG_AUDIT_TREE) += audit_tree.o
obj-$(CONFIG_KPROBES) += kprobes.o
obj-$(CONFIG_UTS_NS) += utsname.o
obj-$(CONFIG_USER_NS) += user_namespace.o
+obj-$(CONFIG_PID_NS) += pid_namespace.o
obj-$(CONFIG_SYSFS) += ksysfs.o
obj-$(CONFIG_DETECT_SOFTLOCKUP) += softlockup.o
obj-$(CONFIG_GENERIC_HARDIRQS) += irq/
diff --git a/kernel/pid.c b/kernel/pid.c
index f815455..21f027c 100644
--- a/kernel/pid.c
+++ b/kernel/pid.c
@@ -41,7 +41,6 @@
static struct hlist_head *pid_hash;
static int pidhash_shift;
struct pid init_struct_pid = INIT_STRUCT_PID;
-static struct kmem_cache *pid_ns_cachep;

int pid_max = PID_MAX_DEFAULT;

@@ -181,7 +180,7 @@ static int alloc_pidmap(struct pid_namespace *pid_ns)
return -1;
}

-static int next_pidmap(struct pid_namespace *pid_ns, int last)
+int next_pidmap(struct pid_namespace *pid_ns, int last)
{
int offset;
struct pidmap *map, *end;
@@ -487,180 +486,6 @@ struct pid *find_get_pid(int nr, struct pid_namespace *ns)
}
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,
- sizeof(struct pid) + (nr_ids - 1) * sizeof(struct upid),
- 0, SLAB_HWCACHE_ALIGN, 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;
-}
-
-#ifdef CONFIG_PID_NS
-static struct pid_namespace *create_pid_namespace(int level)
-{
- struct pid_namespace *ns;
- int i;
-
- ns = kmem_cache_alloc(pid_ns_cachep, GFP_KERNEL);
- if (ns == NULL)
- goto out;
-
- ns->pidmap[0].page = kzalloc(PAGE_SIZE, GFP_KERNEL);

```

```

- if (!ns->pidmap[0].page)
- goto out_free;
-
- ns->pid_cachep = create_pid_cachep(level + 1);
- if (ns->pid_cachep == NULL)
- goto out_free_map;
-
- kref_init(&ns->kref);
- ns->last_pid = 0;
- ns->child_reaper = NULL;
- ns->level = level;
-
- set_bit(0, ns->pidmap[0].page);
- atomic_set(&ns->pidmap[0].nr_free, BITS_PER_PAGE - 1);
-
- for (i = 1; i < PIDMAP_ENTRIES; i++) {
- ns->pidmap[i].page = 0;
- atomic_set(&ns->pidmap[i].nr_free, BITS_PER_PAGE);
- }
-
- return ns;
-
-out_free_map:
- kfree(ns->pidmap[0].page);
-out_free:
- kmem_cache_free(pid_ns_cachep, ns);
-out:
- return ERR_PTR(-ENOMEM);
-}
-
-static void destroy_pid_namespace(struct pid_namespace *ns)
-{
- int i;
-
- for (i = 0; i < PIDMAP_ENTRIES; i++)
- kfree(ns->pidmap[i].page);
- kmem_cache_free(pid_ns_cachep, ns);
-}
-
-struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *old_ns)
-{
- struct pid_namespace *new_ns;
-
- BUG_ON(!old_ns);
- new_ns = get_pid_ns(old_ns);
- if (!(flags & CLONE_NEWPID))
- goto out;
-
-

```

```

- new_ns = ERR_PTR(-EINVAL);
- if (flags & CLONE_THREAD)
- goto out_put;
-
- new_ns = create_pid_namespace(old_ns->level + 1);
- if (!IS_ERR(new_ns))
- new_ns->parent = get_pid_ns(old_ns);
-
-out_put:
- put_pid_ns(old_ns);
-out:
- return new_ns;
-}
-
-void free_pid_ns(struct kref *kref)
-{
- struct pid_namespace *ns, *parent;
-
- ns = container_of(kref, struct pid_namespace, kref);
-
- parent = ns->parent;
- destroy_pid_namespace(ns);
-
- if (parent != NULL)
- put_pid_ns(parent);
-}
-#endif /* CONFIG_PID_NS */
-
-void zap_pid_ns_processes(struct pid_namespace *pid_ns)
-{
- int nr;
- int rc;
-
- /*
-  * The last thread in the cgroup-init thread group is terminating.
-  * Find remaining pid_ts in the namespace, signal and wait for them
-  * to exit.
-  *
-  * Note: This signals each threads in the namespace - even those that
-  * belong to the same thread group, To avoid this, we would have
-  * to walk the entire tasklist looking a processes in this
-  * namespace, but that could be unnecessarily expensive if the
-  * pid namespace has just a few processes. Or we need to
-  * maintain a tasklist for each pid namespace.
-  *
-  */
- read_lock(&tasklist_lock);
- nr = next_pidmap(pid_ns, 1);

```

```

- while (nr > 0) {
-   kill_proc_info(SIGKILL, SEND_SIG_PRIV, nr);
-   nr = next_pidmap(pid_ns, nr);
- }
- read_unlock(&tasklist_lock);
-
- do {
-   clear_thread_flag(TIF_SIGPENDING);
-   rc = sys_wait4(-1, NULL, __WALL, NULL);
- } while (rc != -ECHILD);
-
-
- /* Child reaper for the pid namespace is going away */
- pid_ns->child_reaper = NULL;
- return;
-}
-
-/*
- * 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
-@@ -693,9 +518,6 @@ void __init pidmap_init(void)
-   set_bit(0, init_pid_ns.pidmap[0].page);
-   atomic_dec(&init_pid_ns.pidmap[0].nr_free);
-
-   init_pid_ns.pid_cachep = create_pid_cachep(1);
-   if (init_pid_ns.pid_cachep == NULL)
-   panic("Can't create pid_1 cachep\n");
-
-   pid_ns_cachep = KMEM_CACHE(pid_namespace, SLAB_PANIC);
-+ init_pid_ns.pid_cachep = KMEM_CACHE(pid,
-+   SLAB_HWCACHE_ALIGN | SLAB_PANIC);
-}
diff --git a/kernel/pid_namespace.c b/kernel/pid_namespace.c
new file mode 100644
index 0000000..6d8c859
--- /dev/null
+++ b/kernel/pid_namespace.c
@@ -0,0 +1,197 @@
+/*
+ * Pid namespaces
+ *
+ * Authors:
+ *   (C) 2007 Pavel Emelyanov <xemul@openvz.org>, OpenVZ, SWsoft Inc.
+ *   (C) 2007 Sukadev Bhattiprolu <sukadev@us.ibm.com>, IBM
+ *   Many thanks to Oleg Nesterov for comments and help
+ *
+ */
+

```

```

+#include <linux/pid.h>
+#include <linux/pid_namespace.h>
+#include <linux/syscalls.h>
+#include <linux/err.h>
+
+#define BITS_PER_PAGE (PAGE_SIZE*8)
+
+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);
+static struct kmem_cache *pid_ns_cachep;
+
+/*
+ * 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,
+ sizeof(struct pid) + (nr_ids - 1) * sizeof(struct upid),
+ 0, SLAB_HWCACHE_ALIGN, 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;
+}
+
+static struct pid_namespace *create_pid_namespace(int level)
+{
+ struct pid_namespace *ns;
+ int i;
+
+ ns = kmem_cache_alloc(pid_ns_cachep, GFP_KERNEL);
+ if (ns == NULL)
+ goto out;
+
+ ns->pidmap[0].page = kzalloc(PAGE_SIZE, GFP_KERNEL);
+ if (!ns->pidmap[0].page)
+ goto out_free;
+
+ ns->pid_cachep = create_pid_cachep(level + 1);
+ if (ns->pid_cachep == NULL)
+ goto out_free_map;
+
+ kref_init(&ns->kref);
+ ns->last_pid = 0;
+ ns->child_reaper = NULL;
+ ns->level = level;
+
+ set_bit(0, ns->pidmap[0].page);
+ atomic_set(&ns->pidmap[0].nr_free, BITS_PER_PAGE - 1);
+
+ for (i = 1; i < PIDMAP_ENTRIES; i++) {
+ ns->pidmap[i].page = 0;
+ atomic_set(&ns->pidmap[i].nr_free, BITS_PER_PAGE);
+ }
+
+ return ns;
+
+out_free_map:
+ kfree(ns->pidmap[0].page);
+out_free:
+ kmem_cache_free(pid_ns_cachep, ns);
+out:
+ return ERR_PTR(-ENOMEM);

```

```

+}
+
+static void destroy_pid_namespace(struct pid_namespace *ns)
+{
+ int i;
+
+ for (i = 0; i < PIDMAP_ENTRIES; i++)
+ kfree(ns->pidmap[i].page);
+ kmem_cache_free(pid_ns_cachep, ns);
+}
+
+struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *old_ns)
+{
+ struct pid_namespace *new_ns;
+
+ BUG_ON(!old_ns);
+ new_ns = get_pid_ns(old_ns);
+ if (!(flags & CLONE_NEWPID))
+ goto out;
+
+ new_ns = ERR_PTR(-EINVAL);
+ if (flags & CLONE_THREAD)
+ goto out_put;
+
+ new_ns = create_pid_namespace(old_ns->level + 1);
+ if (!IS_ERR(new_ns))
+ new_ns->parent = get_pid_ns(old_ns);
+
+out_put:
+ put_pid_ns(old_ns);
+out:
+ return new_ns;
+}
+
+void free_pid_ns(struct kref *kref)
+{
+ struct pid_namespace *ns, *parent;
+
+ ns = container_of(kref, struct pid_namespace, kref);
+
+ parent = ns->parent;
+ destroy_pid_namespace(ns);
+
+ if (parent != NULL)
+ put_pid_ns(parent);
+}
+
+void zap_pid_ns_processes(struct pid_namespace *pid_ns)

```

```

+{
+ int nr;
+ int rc;
+
+ /*
+  * The last thread in the cgroup-init thread group is terminating.
+  * Find remaining pid_ts in the namespace, signal and wait for them
+  * to exit.
+  *
+  * Note: This signals each threads in the namespace - even those that
+  * belong to the same thread group, To avoid this, we would have
+  * to walk the entire tasklist looking a processes in this
+  * namespace, but that could be unnecessarily expensive if the
+  * pid namespace has just a few processes. Or we need to
+  * maintain a tasklist for each pid namespace.
+  */
+ read_lock(&tasklist_lock);
+ nr = next_pidmap(pid_ns, 1);
+ while (nr > 0) {
+ kill_proc_info(SIGKILL, SEND_SIG_PRIV, nr);
+ nr = next_pidmap(pid_ns, nr);
+ }
+ read_unlock(&tasklist_lock);
+
+ do {
+ clear_thread_flag(TIF_SIGPENDING);
+ rc = sys_wait4(-1, NULL, __WALL, NULL);
+ } while (rc != -ECHILD);
+
+
+ /* Child reaper for the pid namespace is going away */
+ pid_ns->child_reaper = NULL;
+ return;
+}
+
+static __init int pid_namespaces_init(void)
+{
+ pid_ns_cachep = KMEM_CACHE(pid_namespace, SLAB_PANIC);
+ return 0;
+}
+
+__initcall(pid_namespaces_init);

```

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