Subject: [PATCH 0/6] A config option to compile out some namespaces code (v4)
Posted by Pavel Emelianov on Wed, 14 Nov 2007 11:26:19 GMT
View Forum Message <> Reply to Message

Change against v3: rebased on 2.6.24-rc2-mm1

There were some questions like "do I need this on my cellphone"
in reply to different namespaces patches. Indeed, the namespaces
are not useful for most of the embedded systems, but the code
creating and releasing them weights a lot.

So I propose to add a config option which will help embedded
people to reduce the vmlinux size. This option simply compiles
out the namespaces cloning and releasing code *only*, but keeps
all the other logic untouched (e.g. the notion of init_ns).

Moreover, some of the namespaces might be not 100% ready by
the time of Linux-2.6.xxx release (like user namespaces or pid
namespaces are now). Since each namespace has its own option,
which depends on the NAMESPACES, it can be mrked with "depends
on EXPERIMENTAL/BROKEN/ANYTHING_ELSE" not to release the
functionality that is not 100% ready yet.

When someone tries to clone some namespace with their support
turned off, he will receive an EINVAL error.

This patchset can save more than 2KB from the vmlinux when
turning the config option "NAMESPACES" to "n".

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

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

Subject: [PATCH 1/6] Add the NAMESPACES config option
Posted by Pavel Emelianov on Wed, 14 Nov 2007 11:28:17 GMT
View Forum Message <> Reply to Message

The option is selectable if EMBEDDED is chosen only. When
the EMBEDDED is off namespaces will be on.

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

---

diff --git a/init/Kconfig b/init/Kconfig

```
index 96fba82..4ccc1a0 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -404,6 +404,15 @@ config RELAY

    If unsure, say N.

+config NAMESPACES
+ bool "Namespaces support" if EMBEDDED
+ default !EMBEDDED
+ help
+   Provides the way to make tasks work with different objects using
+   the same id. For example same IPC id may refer to different objects
+   or same user id or pid may refer to different tasks when used in
+   different namespaces.
+
 config BLK_DEV_INITRD
  bool "Initial RAM filesystem and RAM disk (initramfs/initrd) support"
  depends on BROKEN || !FRV
--
1.5.3.4
```

Subject: [PATCH 2/6] Move the UTS namespace under UTS_NS option
Posted by Pavel Emelianov on Wed, 14 Nov 2007 11:30:06 GMT
View Forum Message <> Reply to Message

Currently all the namespace management code is in the
kernel/utsname.c file, so just compile it out and make stubs
in the appropriate header.

The init namespace itself is in init/version.c and is in the
kernel all the time.

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

```
diff --git a/include/linux/utsname.h b/include/linux/utsname.h
index 923db99..1123267 100644
--- a/include/linux/utsname.h
+++ b/include/linux/utsname.h
@@ -35,6 +35,7 @@ struct new_utsname {
```

```
 #include <linux/sched.h>
 #include <linux/kref.h>
 #include <linux/nsproxy.h>
+#include <linux/err.h>
 #include <asm/atomic.h>

 struct uts_namespace {
@@ -43,6 +44,7 @@ struct uts_namespace {
 };
 extern struct uts_namespace init_uts_ns;

+#ifdef CONFIG_UTS_NS
 static inline void get_uts_ns(struct uts_namespace *ns)
 {
  kref_get(&ns->kref);
@@ -56,6 +58,25 @@ static inline void put_uts_ns(struct uts_namespace *ns)
 {
  kref_put(&ns->kref, free_uts_ns);
 }
+#else
+static inline void get_uts_ns(struct uts_namespace *ns)
+{
+}
+
+static inline void put_uts_ns(struct uts_namespace *ns)
+{
+}
+
+static inline struct uts_namespace *copy_utsname(unsigned long flags,
+     struct uts_namespace *ns)
+{
+ if (flags & CLONE_NEWUTS)
+  return ERR_PTR(-EINVAL);
+
+ return ns;
+}
+#endif
+
 static inline struct new_utsname *utsname(void)
 {
  return &current->nsproxy->uts_ns->name;
diff --git a/init/Kconfig b/init/Kconfig
index 4ccc1a0..2139218 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -413,6 +413,13 @@ config NAMESPACES
    or same user id or pid may refer to different tasks when used in
    different namespaces.
```

```
+config UTS_NS
+ bool "UTS namespace"
+ depends on NAMESPACES
+ help
+   In this namespace tasks see different info provided with the
+   uname() system call
+
 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 876dbcd..19d2411 100644
--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ -9,7 +9,7 @@ obj-y     = sched.o fork.o exec_domain.o panic.o printk.o profile.o \
       rcupdate.o extable.o params.o posix-timers.o \
       kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
       hrtimer.o rwsem.o nsproxy.o srcu.o \
-      utsname.o notifier.o pm_qos_params.o
+      notifier.o pm_qos_params.o

 obj-$(CONFIG_SYSCTL) += sysctl_check.o
 obj-$(CONFIG_STACKTRACE) += stacktrace.o
@@ -48,6 +48,7 @@ obj-$(CONFIG_AUDIT) += audit.o auditfilter.o
 obj-$(CONFIG_AUDITSYSCALL) += auditsc.o
 obj-$(CONFIG_AUDIT_TREE) += audit_tree.o
 obj-$(CONFIG_KPROBES) += kprobes.o
+obj-$(CONFIG_UTS_NS) += utsname.o
 obj-$(CONFIG_SYSFS) += ksysfs.o
 obj-$(CONFIG_DETECT_SOFTLOCKUP) += softlockup.o
 obj-$(CONFIG_GENERIC_HARDIRQS) += irq/
--
1.5.3.4
```


Subject: [PATCH 3/6] Move the IPC namespace under IPC_NS option
Posted by Pavel Emelianov on Wed, 14 Nov 2007 11:32:43 GMT
View Forum Message <> Reply to Message

Currently the IPC namespace management code is spread over
the ipc/*.c files. I moved this code into ipc/namespace.c file
which is compiled out when needed.

The linux/ipc_namespace.h file is used to store the
prototypes of the functions in namespace.c and the stubs
for NAMESPACES=n case. This is done so, because the stub
for copy_ipc_namespace requires the knowledge of the
CLONE_NEWIPC flag, which is in sched.h. But the linux/ipc.h
file itself in included into many many .c files via the
sys.h->sem.h sequence so adding the sched.h into it will
make all these .c depend on sched.h which is not that good.
On the other hand the knowledge about the namespaces stuff
is required in 4 .c files only.

Besides, this patch compiles out some auxiliary functions
from ipc/sem.c, msg.c and shm.c files. It turned out that
moving these functions into namespaces.c is not that easy
because they use many other calls and macros from the original
file. Moving them would make this patch complicated. On the
other hand all these functions can be consolidated, so I
will send a separate patch doing this a bit later.

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

```
diff --git a/include/linux/ipc.h b/include/linux/ipc.h
index 408696e..b882610 100644
--- a/include/linux/ipc.h
+++ b/include/linux/ipc.h
@@ -100,58 +100,6 @@ struct kern_ipc_perm
  void  *security;
 };

-struct ipc_ids;
-struct ipc_namespace {
- struct kref kref;
- struct ipc_ids *ids[3];
-
- int  sem_ctls[4];
- int  used_sems;
-
- int  msg_ctlmax;
- int  msg_ctlmnb;
- int  msg_ctlmni;
- atomic_t msg_bytes;
- atomic_t msg_hdrs;
-
- size_t  shm_ctlmax;
- size_t  shm_ctlall;
```

```
- int  shm_ctlmni;
- int  shm_tot;
-};
-
-extern struct ipc_namespace init_ipc_ns;
-
-#ifdef CONFIG_SYSVIPC
-#define INIT_IPC_NS(ns)  .ns  = &init_ipc_ns,
-extern void free_ipc_ns(struct kref *kref);
-extern struct ipc_namespace *copy_ipcs(unsigned long flags,
-      struct ipc_namespace *ns);
-#else
-#define INIT_IPC_NS(ns)
-static inline struct ipc_namespace *copy_ipcs(unsigned long flags,
-      struct ipc_namespace *ns)
-{
- return ns;
-}
-#endif
-
-static inline struct ipc_namespace *get_ipc_ns(struct ipc_namespace *ns)
-{
-#ifdef CONFIG_SYSVIPC
- if (ns)
-  kref_get(&ns->kref);
-#endif
- return ns;
-}
-
-static inline void put_ipc_ns(struct ipc_namespace *ns)
-{
-#ifdef CONFIG_SYSVIPC
- kref_put(&ns->kref, free_ipc_ns);
-#endif
-}
-
 #endif /* __KERNEL__ */

 #endif /* _LINUX_IPC_H */
diff --git a/include/linux/ipc_namespace.h b/include/linux/ipc_namespace.h
new file mode 100644
index 0000000..a491fc9
--- /dev/null
+++ b/include/linux/ipc_namespace.h
@@ -0,0 +1,69 @@
+#ifndef __IPC_NAMESPACE_H__
+#define __IPC_NAMESPACE_H__
+
```

```
+#include <linux/err.h>
+
+struct ipc_ids;
+struct ipc_namespace {
+ struct kref kref;
+ struct ipc_ids *ids[3];
+
+ int  sem_ctls[4];
+ int  used_sems;
+
+ int  msg_ctlmax;
+ int  msg_ctlmnb;
+ int  msg_ctlmni;
+ atomic_t msg_bytes;
+ atomic_t msg_hdrs;
+
+ size_t  shm_ctlmax;
+ size_t  shm_ctlall;
+ int  shm_ctlmni;
+ int  shm_tot;
+};
+
+extern struct ipc_namespace init_ipc_ns;
+
+#ifdef CONFIG_SYSVIPC
+#define INIT_IPC_NS(ns)  .ns  = &init_ipc_ns,
+#else
+#define INIT_IPC_NS(ns)
+#endif
+
+#if defined(CONFIG_SYSVIPC) && defined(CONFIG_IPC_NS)
+extern void free_ipc_ns(struct kref *kref);
+extern struct ipc_namespace *copy_ipcs(unsigned long flags,
+      struct ipc_namespace *ns);
+
+static inline struct ipc_namespace *get_ipc_ns(struct ipc_namespace *ns)
+{
+ if (ns)
+  kref_get(&ns->kref);
+ return ns;
+}
+
+static inline void put_ipc_ns(struct ipc_namespace *ns)
+{
+ kref_put(&ns->kref, free_ipc_ns);
+}
+#else
+static inline struct ipc_namespace *copy_ipcs(unsigned long flags,
```

```
+  struct ipc_namespace *ns)
+{
+ if (flags & CLONE_NEWIPC)
+  return ERR_PTR(-EINVAL);
+
+ return ns;
+}
+
+static inline struct ipc_namespace *get_ipc_ns(struct ipc_namespace *ns)
+{
+ return ns;
+}
+
+static inline void put_ipc_ns(struct ipc_namespace *ns)
+{
+}
+#endif
+#endif
diff --git a/init/Kconfig b/init/Kconfig
index 2139218..fe73fe6 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -420,6 +420,13 @@ config UTS_NS
    In this namespace tasks see different info provided with the
    uname() system call

+config IPC_NS
+ bool "IPC namespace"
+ depends on NAMESPACES && SYSVIPC
+ help
+   In this namespace tasks work with IPC ids which correspond to
+   different IPC objects in different namespaces
+
 config BLK_DEV_INITRD
  bool "Initial RAM filesystem and RAM disk (initramfs/initrd) support"
  depends on BROKEN || !FRV
diff --git a/ipc/Makefile b/ipc/Makefile
index b93bba6..5fc5e33 100644
--- a/ipc/Makefile
+++ b/ipc/Makefile
@@ -7,4 +7,5 @@ obj-$(CONFIG_SYSVIPC) += util.o msgutil.o msg.o sem.o shm.o
 obj-$(CONFIG_SYSVIPC_SYSCTL) += ipc_sysctl.o
 obj_mq-$(CONFIG_COMPAT) += compat_mq.o
 obj-$(CONFIG_POSIX_MQUEUE) += mqueue.o msgutil.o $(obj_mq-y)
+obj-$(CONFIG_IPC_NS) += namespace.o

diff --git a/ipc/ipc_sysctl.c b/ipc/ipc_sysctl.c
index 79e24e8..7f4235b 100644
```

```
--- a/ipc/ipc_sysctl.c
+++ b/ipc/ipc_sysctl.c
@@ -14,6 +14,7 @@
 #include <linux/nsproxy.h>
 #include <linux/sysctl.h>
 #include <linux/uaccess.h>
+#include <linux/ipc_namespace.h>

 static void *get_ipc(ctl_table *table)
 {
diff --git a/ipc/msg.c b/ipc/msg.c
index ec0c724..5879bfe 100644
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -36,6 +36,7 @@
 #include <linux/seq_file.h>
 #include <linux/rwsem.h>
 #include <linux/nsproxy.h>
+#include <linux/ipc_namespace.h>

 #include <asm/current.h>
 #include <asm/uaccess.h>
@@ -90,6 +91,7 @@ static void __msg_init_ns(struct ipc_namespace *ns, struct ipc_ids *ids)
  ipc_init_ids(ids);
 }

+#ifdef CONFIG_IPC_NS
 int msg_init_ns(struct ipc_namespace *ns)
 {
  struct ipc_ids *ids;
@@ -128,6 +130,7 @@ void msg_exit_ns(struct ipc_namespace *ns)
  kfree(ns->ids[IPC_MSG_IDS]);
  ns->ids[IPC_MSG_IDS] = NULL;
 }
+#endif

 void __init msg_init(void)
 {
diff --git a/ipc/namespace.c b/ipc/namespace.c
new file mode 100644
index 0000000..cef1139
--- /dev/null
+++ b/ipc/namespace.c
@@ -0,0 +1,73 @@
+/*
+ * linux/ipc/namespace.c
+ * Copyright (C) 2006 Pavel Emelyanov <xemul@openvz.org> OpenVZ, SWsoft Inc.
+ */
```

```
+
+#include <linux/ipc.h>
+#include <linux/msg.h>
+#include <linux/ipc_namespace.h>
+#include <linux/rcupdate.h>
+#include <linux/nsproxy.h>
+#include <linux/slab.h>
+
+#include "util.h"
+
+static struct ipc_namespace *clone_ipc_ns(struct ipc_namespace *old_ns)
+{
+ int err;
+ struct ipc_namespace *ns;
+
+ err = -ENOMEM;
+ ns = kmalloc(sizeof(struct ipc_namespace), GFP_KERNEL);
+ if (ns == NULL)
+  goto err_mem;
+
+ err = sem_init_ns(ns);
+ if (err)
+  goto err_sem;
+ err = msg_init_ns(ns);
+ if (err)
+  goto err_msg;
+ err = shm_init_ns(ns);
+ if (err)
+  goto err_shm;
+
+ kref_init(&ns->kref);
+ return ns;
+
+err_shm:
+ msg_exit_ns(ns);
+err_msg:
+ sem_exit_ns(ns);
+err_sem:
+ kfree(ns);
+err_mem:
+ return ERR_PTR(err);
+}
+
+struct ipc_namespace *copy_ipcs(unsigned long flags, struct ipc_namespace *ns)
+{
+ struct ipc_namespace *new_ns;
+
+ BUG_ON(!ns);
```

```
+ get_ipc_ns(ns);
+
+ if (!(flags & CLONE_NEWIPC))
+  return ns;
+
+ new_ns = clone_ipc_ns(ns);
+
+ put_ipc_ns(ns);
+ return new_ns;
+}
+
+void free_ipc_ns(struct kref *kref)
+{
+ struct ipc_namespace *ns;
+
+ ns = container_of(kref, struct ipc_namespace, kref);
+ sem_exit_ns(ns);
+ msg_exit_ns(ns);
+ shm_exit_ns(ns);
+ kfree(ns);
+}
diff --git a/ipc/sem.c b/ipc/sem.c
index d65e285..84c701f 100644
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -82,6 +82,7 @@
 #include <linux/seq_file.h>
 #include <linux/rwsem.h>
 #include <linux/nsproxy.h>
+#include <linux/ipc_namespace.h>

 #include <asm/uaccess.h>
 #include "util.h"
@@ -128,6 +129,7 @@ static void __sem_init_ns(struct ipc_namespace *ns, struct ipc_ids *ids)
  ipc_init_ids(ids);
 }

+#ifdef CONFIG_IPC_NS
 int sem_init_ns(struct ipc_namespace *ns)
 {
  struct ipc_ids *ids;
@@ -165,6 +167,7 @@ void sem_exit_ns(struct ipc_namespace *ns)
  kfree(ns->ids[IPC_SEM_IDS]);
  ns->ids[IPC_SEM_IDS] = NULL;
 }
+#endif

 void __init sem_init (void)
```

```
 {
diff --git a/ipc/shm.c b/ipc/shm.c
index 65c3a29..07f4b7a 100644
--- a/ipc/shm.c
+++ b/ipc/shm.c
@@ -38,6 +38,7 @@
 #include <linux/rwsem.h>
 #include <linux/nsproxy.h>
 #include <linux/mount.h>
+#include <linux/ipc_namespace.h>

 #include <asm/uaccess.h>

@@ -96,6 +97,7 @@ static void do_shm_rmid(struct ipc_namespace *ns, struct shmid_kernel
*shp)
   shm_destroy(ns, shp);
 }

+#ifdef CONFIG_IPC_NS
 int shm_init_ns(struct ipc_namespace *ns)
 {
  struct ipc_ids *ids;
@@ -133,6 +135,7 @@ void shm_exit_ns(struct ipc_namespace *ns)
  kfree(ns->ids[IPC_SHM_IDS]);
  ns->ids[IPC_SHM_IDS] = NULL;
 }
+#endif

 void __init shm_init (void)
 {
diff --git a/ipc/util.c b/ipc/util.c
index 76c1f34..5432b8e 100644
--- a/ipc/util.c
+++ b/ipc/util.c
@@ -33,6 +33,7 @@
 #include <linux/audit.h>
 #include <linux/nsproxy.h>
 #include <linux/rwsem.h>
+#include <linux/ipc_namespace.h>

 #include <asm/unistd.h>

@@ -51,66 +52,6 @@ struct ipc_namespace init_ipc_ns = {
  },
 };

-static struct ipc_namespace *clone_ipc_ns(struct ipc_namespace *old_ns)
-{
```

```
- int err;
- struct ipc_namespace *ns;
-
- err = -ENOMEM;
- ns = kmalloc(sizeof(struct ipc_namespace), GFP_KERNEL);
- if (ns == NULL)
-  goto err_mem;
-
- err = sem_init_ns(ns);
- if (err)
-  goto err_sem;
- err = msg_init_ns(ns);
- if (err)
-  goto err_msg;
- err = shm_init_ns(ns);
- if (err)
-  goto err_shm;
-
- kref_init(&ns->kref);
- return ns;
-
-err_shm:
- msg_exit_ns(ns);
-err_msg:
- sem_exit_ns(ns);
-err_sem:
- kfree(ns);
-err_mem:
- return ERR_PTR(err);
-}
-
-struct ipc_namespace *copy_ipcs(unsigned long flags, struct ipc_namespace *ns)
-{
- struct ipc_namespace *new_ns;
-
- BUG_ON(!ns);
- get_ipc_ns(ns);
-
- if (!(flags & CLONE_NEWIPC))
-  return ns;
-
- new_ns = clone_ipc_ns(ns);
-
- put_ipc_ns(ns);
- return new_ns;
-}
-
-void free_ipc_ns(struct kref *kref)
```

```
-{
- struct ipc_namespace *ns;
-
- ns = container_of(kref, struct ipc_namespace, kref);
- sem_exit_ns(ns);
- msg_exit_ns(ns);
- shm_exit_ns(ns);
- kfree(ns);
-}
-
 /**
  * ipc_init - initialise IPC subsystem
  *
diff --git a/ipc/util.h b/ipc/util.h
index 9ffea40..fc6b729 100644
--- a/ipc/util.h
+++ b/ipc/util.h
@@ -20,6 +20,8 @@ void sem_init (void);
 void msg_init (void);
 void shm_init (void);

+struct ipc_namespace;
+
 int sem_init_ns(struct ipc_namespace *ns);
 int msg_init_ns(struct ipc_namespace *ns);
 int shm_init_ns(struct ipc_namespace *ns);
diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index 79f871b..f5d332c 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -21,6 +21,7 @@
 #include <linux/utsname.h>
 #include <linux/pid_namespace.h>
 #include <net/net_namespace.h>
+#include <linux/ipc_namespace.h>

 static struct kmem_cache *nsproxy_cachep;

--
1.5.3.4
```

Subject: [PATCH 4/6] Cleanup the code managed with the USER_NS option
Posted by Pavel Emelianov on Wed, 14 Nov 2007 11:36:57 GMT
View Forum Message <> Reply to Message

Make the user_namespace.o compilation depend on this option and
move the init_user_ns into user.c file to make the kernel compile
and work without the namespaces support. This make the user
namespace code be organized similar to other namespaces'.

Also mask the USER_NS option as "depend on NAMESPACES".

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

```
diff --git a/init/Kconfig b/init/Kconfig
index fe73fe6..825f10c 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -206,15 +206,6 @@ config TASK_IO_ACCOUNTING

    Say N if unsure.

-config USER_NS
- bool "User Namespaces (EXPERIMENTAL)"
- default n
- depends on EXPERIMENTAL
- help
-   Support user namespaces.  This allows containers, i.e.
-   vservers, to use user namespaces to provide different
-   user info for different servers.  If unsure, say N.
-
 config PID_NS
  bool "PID Namespaces (EXPERIMENTAL)"
  default n
@@ -427,6 +418,14 @@ config IPC_NS
    In this namespace tasks work with IPC ids which correspond to
    different IPC objects in different namespaces

+config USER_NS
+ bool "User namespace (EXPERIMENTAL)"
+ depends on NAMESPACES && EXPERIMENTAL
+ help
+   This allows containers, i.e. vservers, to use user namespaces
+   to provide different user info for different servers.
+   If unsure, say N.
+
 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 19d2411..d01cb7b 100644
--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ -4,7 +4,7 @@

 obj-y     = sched.o fork.o exec_domain.o panic.o printk.o profile.o \
       exit.o itimer.o time.o softirq.o resource.o \
-      sysctl.o capability.o ptrace.o timer.o user.o user_namespace.o \
+      sysctl.o capability.o ptrace.o timer.o user.o \
       signal.o sys.o kmod.o workqueue.o pid.o \
       rcupdate.o extable.o params.o posix-timers.o \
       kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
@@ -49,6 +49,7 @@ obj-$(CONFIG_AUDITSYSCALL) += auditsc.o
 obj-$(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_SYSFS) += ksysfs.o
 obj-$(CONFIG_DETECT_SOFTLOCKUP) += softlockup.o
 obj-$(CONFIG_GENERIC_HARDIRQS) += irq/
diff --git a/kernel/user.c b/kernel/user.c
index 74a1ddf..624e3d7 100644
--- a/kernel/user.c
+++ b/kernel/user.c
@@ -17,6 +17,15 @@
 #include <linux/module.h>
 #include <linux/user_namespace.h>

+struct user_namespace init_user_ns = {
+ .kref = {
+  .refcount = ATOMIC_INIT(2),
+ },
+ .root_user = &root_user,
+};
+
+EXPORT_SYMBOL_GPL(init_user_ns);
+
 /*
  * UID task count cache, to get fast user lookup in "alloc_uid"
  * when changing user ID's (ie setuid() and friends).
@@ -430,6 +439,7 @@ void switch_uid(struct user_struct *new_user)
  suid_keys(current);
 }

+#ifdef CONFIG_USER_NS
 void release_uids(struct user_namespace *ns)
```

```
 {
  int i;
@@ -454,6 +464,7 @@ void release_uids(struct user_namespace *ns)

  free_uid(ns->root_user);
 }
+#endif

 static int __init uid_cache_init(void)
 {
diff --git a/kernel/user_namespace.c b/kernel/user_namespace.c
index 7af90fc..4c90062 100644
--- a/kernel/user_namespace.c
+++ b/kernel/user_namespace.c
@@ -10,17 +10,6 @@
 #include <linux/nsproxy.h>
 #include <linux/user_namespace.h>

-struct user_namespace init_user_ns = {
- .kref = {
-  .refcount = ATOMIC_INIT(2),
- },
- .root_user = &root_user,
-};
-
-EXPORT_SYMBOL_GPL(init_user_ns);
-
-#ifdef CONFIG_USER_NS
-
 /*
  * Clone a new ns copying an original user ns, setting refcount to 1
  * @old_ns: namespace to clone
@@ -84,5 +73,3 @@ void free_user_ns(struct kref *kref)
  release_uids(ns);
  kfree(ns);
 }
-
-#endif /* CONFIG_USER_NS */
--
1.5.3.4
```

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_ge_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
-   Suport 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
+   Suport 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_ge_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);
```

Subject: [PATCH 6/6] Mark NET_NS with "depends on NAMESPACES"
Posted by Pavel Emelianov on Wed, 14 Nov 2007 11:41:14 GMT
View Forum Message <> Reply to Message

There's already an option controlling the net namespaces
cloning code, so make it work the same way as all the
other namespaces' options.

Should I wait till the option itself gets to mainline and
resend this patch to David?

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

---

diff --git a/net/Kconfig b/net/Kconfig
index ab4e6da..d61a025 100644
--- a/net/Kconfig
+++ b/net/Kconfig
@@ -30,7 +30,7 @@ menu "Networking options"
 config NET_NS
  bool "Network namespace support"
  default n
- depends on EXPERIMENTAL && !SYSFS
+ depends on EXPERIMENTAL && !SYSFS && NAMESPACES
  help
    Allow user space to create what appear to be multiple instances
    of the network stack.

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

Subject: Re: [PATCH 0/6] A config option to compile out some namespaces code (v4)
Posted by serue on Wed, 14 Nov 2007 14:58:46 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> Change against v3: rebased on 2.6.24-rc2-mm1
>
> There were some questions like "do I need this on my cellphone"
> in reply to different namespaces patches. Indeed, the namespaces
> are not useful for most of the embedded systems, but the code
> creating and releasing them weights a lot.
>
> So I propose to add a config option which will help embedded

> people to reduce the vmlinux size. This option simply compiles
> out the namespaces cloning and releasing code *only*, but keeps
> all the other logic untouched (e.g. the notion of init_ns).
>
> Moreover, some of the namespaces might be not 100% ready by
> the time of Linux-2.6.xxx release (like user namespaces or pid
> namespaces are now). Since each namespace has its own option,
> which depends on the NAMESPACES, it can be mrked with "depends
> on EXPERIMENTAL/BROKEN/ANYTHING_ELSE" not to release the
> functionality that is not 100% ready yet.
>
> When someone tries to clone some namespace with their support
> turned off, he will receive an EINVAL error.
>
> This patchset can save more than 2KB from the vmlinux when
> turning the config option "NAMESPACES" to "n".
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Pretty sure I did this a few versions ago, but

Acked-by: Serge Hallyn <serue@us.ibm.com>

for the whole set.

thanks,
-serge

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