Subject: [PATCH 1/4] Virtualization/containers: introduction
Posted by Kirill Korotaev on Mon, 06 Feb 2006 21:55:44 GMT
View Forum Message <> Reply to Message

Hello,

I tried to take into account all the comments from you guys (thanks a lot for them!) and prepared a new version of virtualization patches. I will send only 4 patches today, just not to overflow everyone and keep it clear/tidy/possible to review.

This patch introduces some abstract container kernel structure and a number of operations on it.

The important properties of the proposed container implementation:
- each container has unique ID in the system
- each process in the kernel can belong to one container only
- effective container pointer (econtainer()) is used on the task to avoid insertion of additional argument "container" to all functions where it is required.
- kernel compilation with disabled virtualization should result in old good linux kernel

Patches following this one will be used for virtualization of the kernel resources based on this container infrastructure, including those VPID patches I sent before. Every virtualized resource can be given separate config option if needed (just give me to know if it is desired).

Signed-Off-By: Kirill Korotaev <dev@openvz.org>

Kirill

P.S. I understand that this virtualization spam can be unintersting for some of you, just give me to know if you want to be removed from CC.

P.P.S. All patches are against 2.6.16-rc2-git2 and compile when virtualization=n.

```
--- ./include/linux/container.h.vpsinfo 2006-02-06 22:35:36.000000000 +0300
+++ ./include/linux/container.h 2006-02-07 00:52:57.000000000 +0300
@@ -0,0 +1,59 @@
+#ifndef __LINUX_CONTAINER_H__
+#define __LINUX_CONTAINER_H__
+
+#include <linux/config.h>
+#include <asm/types.h>
+#include <asm/atomic.h>
```

```
+
+struct task_struct;
+
+struct container {
+ u32 id;
+ struct task_struct *init_task;
+ atomic_t refcnt;
+};
+
+extern struct container init_container;
+
+#ifdef CONFIG_CONTAINER
+
+static inline struct container *get_container(struct container *cont)
+{
+ atomic_inc(&cont->refcnt);
+ return cont;
+}
+
+static inline void put_container(struct container *cont)
+{
+ atomic_dec(&container->refcnt);
+}
+
+#include <linux/sched.h>
+#include <asm/current.h>
+
+static inline struct container *set_econtainer(struct container *cont)
+{
+ struct container *ret;
+
+ ret = current->econtainer;
+ current->econtainer = cont;
+ return ret;
+}
+
+#define econtainer()  (current->econtainer)
+
+extern void init_containers(void);
+
+#else /* CONFIG_CONTAINER */
+
+#define get_container(cont) (NULL)
+#define put_container(cont) do { } while (0)
+
+#define set_econtainer(cont) ((struct container *)NULL)
+#define econtainer()  (NULL)
+
```

```
+#define init_containers() do { } while (0)
+
+#endif /* CONFIG_CONTAINER */
+
+#endif
--- ./include/linux/init_task.h.vpsinfo 2006-01-03 06:21:10.000000000 +0300
+++ ./include/linux/init_task.h 2006-02-07 00:52:52.000000000 +0300
@@ -3,6 +3,7 @@

 #include <linux/file.h>
 #include <linux/rcupdate.h>
+#include <linux/container.h>

 #define INIT_FDTABLE \
 {       \
@@ -131,5 +132,7 @@ extern struct group_info init_groups;
  LIST_HEAD_INIT(cpu_timers[2]),     \
 }

+#define INIT_CONTAINER(cont)      \
+ .refcnt  = ATOMIC_INIT(1)

 #endif
--- ./include/linux/sched.h.vpsinfo 2006-02-06 22:15:05.000000000 +0300
+++ ./include/linux/sched.h 2006-02-07 00:53:32.000000000 +0300
@@ -687,6 +687,7 @@ static inline void prefetch_stack(struct

 struct audit_context;  /* See audit.c */
 struct mempolicy;
+struct container;

 struct task_struct {
  volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
@@ -846,6 +847,11 @@ struct task_struct {

  struct io_context *io_context;

+#ifdef CONFIG_CONTAINER
+ struct container *container;
+ struct container *econtainer; /* effective container */
+#endif
+
  unsigned long ptrace_message;
  siginfo_t *last_siginfo; /* For ptrace use.  */
 /*
--- ./init/main.c.vpsinfo 2006-02-06 22:15:06.000000000 +0300
+++ ./init/main.c 2006-02-07 00:46:12.000000000 +0300
@@ -47,6 +47,7 @@
```

```
 #include <linux/rmap.h>
 #include <linux/mempolicy.h>
 #include <linux/key.h>
+#include <linux/container.h>

 #include <asm/io.h>
 #include <asm/bugs.h>
@@ -603,6 +604,8 @@ static void __init do_initcalls(void)
  */
 static void __init do_basic_setup(void)
 {
+ init_containers();
+
  /* drivers will send hotplug events */
  init_workqueues();
  usermodehelper_init();
--- ./kernel/Makefile.vpsinfo 2006-02-06 22:15:06.000000000 +0300
+++ ./kernel/Makefile 2006-02-07 00:46:12.000000000 +0300
@@ -34,6 +34,7 @@ obj-$(CONFIG_DETECT_SOFTLOCKUP) += softl
 obj-$(CONFIG_GENERIC_HARDIRQS) += irq/
 obj-$(CONFIG_SECCOMP) += seccomp.o
 obj-$(CONFIG_RCU_TORTURE_TEST) += rcutorture.o
+obj-$(CONFIG_CONTAINER) += container.c

 ifneq ($(CONFIG_SCHED_NO_NO_OMIT_FRAME_POINTER),y)
 # According to Alan Modra <alan@linuxcare.com.au>, the -fno-omit-frame-pointer is
--- ./kernel/container.c.vpsinfo 2006-02-07 00:01:21.000000000 +0300
+++ ./kernel/container.c 2006-02-07 00:46:12.000000000 +0300
@@ -0,0 +1,16 @@
+#include <linux/container.h>
+
+/*
+ * Initial container.
+ *
+ * All tasks and other resources initially belong to it
+ */
+struct container init_container = INIT_CONTAINER(init_container);
+
+EXPORT_SYMBOL(init_container);
+
+void init_containers(void)
+{
+ /* remember who is init in container */
+ init_container.init_task = child_reaper;
+}
--- ./kernel/exit.c.vpsinfo 2006-02-06 22:15:06.000000000 +0300
+++ ./kernel/exit.c 2006-02-07 00:46:12.000000000 +0300
@@ -31,6 +31,7 @@
```

```
 #include <linux/signal.h>
 #include <linux/cn_proc.h>
 #include <linux/mutex.h>
+#include <linux/container.h>

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

  p = leader;
--- ./kernel/fork.c.vpsinfo 2006-02-06 22:15:06.000000000 +0300
+++ ./kernel/fork.c 2006-02-07 00:46:12.000000000 +0300
@@ -44,6 +44,7 @@
 #include <linux/rmap.h>
 #include <linux/acct.h>
 #include <linux/cn_proc.h>
+#include <linux/container.h>

 #include <asm/pgtable.h>
 #include <asm/pgalloc.h>
@@ -1132,6 +1133,7 @@ static task_t *copy_process(unsigned lon
  p->ioprio = current->ioprio;

  SET_LINKS(p);
+ (void)get_container(p->container);
  if (unlikely(p->ptrace & PT_PTRACED))
   __ptrace_link(p, current->parent);
```

Subject: [PATCH 2/4] Virtualization/containers: CONFIG_CONTAINER
Posted by Kirill Korotaev on Mon, 06 Feb 2006 22:10:05 GMT
View Forum Message <> Reply to Message

This patch simply adds CONFIG_CONTAINER option for
virtualization/containerss functionality.
Per-resource config options can be added later if needed.

Signed-Off-By: Kirill Korotaev <dev@openvz.org>

Kirill

```
--- ./arch/alpha/Kconfig.vkconfig 2006-02-06 22:14:50.000000000 +0300
+++ ./arch/alpha/Kconfig 2006-02-06 23:26:35.000000000 +0300
```

```
@@ -621,6 +621,8 @@ source "arch/alpha/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/arm/Kconfig.vkconfig 2006-02-06 22:14:50.000000000 +0300
+++ ./arch/arm/Kconfig 2006-02-06 23:27:06.000000000 +0300
@@ -794,6 +794,8 @@ source "arch/arm/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/arm26/Kconfig.vkconfig 2006-02-06 22:14:51.000000000 +0300
+++ ./arch/arm26/Kconfig 2006-02-06 23:27:14.000000000 +0300
@@ -232,6 +232,8 @@ source "arch/arm26/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/cris/Kconfig.vkconfig 2006-02-06 22:14:51.000000000 +0300
+++ ./arch/cris/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -175,6 +175,8 @@ source "arch/cris/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/frv/Kconfig.vkconfig 2006-02-06 22:14:51.000000000 +0300
+++ ./arch/frv/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -341,6 +341,8 @@ source "arch/frv/Kconfig.debug"

 source "security/Kconfig"
```

```
+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/h8300/Kconfig.vkconfig 2006-02-06 22:14:51.000000000 +0300
+++ ./arch/h8300/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -189,6 +189,8 @@ source "arch/h8300/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/i386/Kconfig.vkconfig 2006-02-06 22:15:14.000000000 +0300
+++ ./arch/i386/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -1072,6 +1072,8 @@ source "arch/i386/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/ia64/Kconfig.vkconfig 2006-01-03 06:21:10.000000000 +0300
+++ ./arch/ia64/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -463,4 +463,6 @@ source "arch/ia64/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"
--- ./arch/m32r/Kconfig.vkconfig 2006-02-06 22:14:52.000000000 +0300
+++ ./arch/m32r/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -377,6 +377,8 @@ source "arch/m32r/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/m68k/Kconfig.vkconfig 2006-02-06 22:14:52.000000000 +0300
```

```
+++ ./arch/m68k/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -648,6 +648,8 @@ source "arch/m68k/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/m68knommu/Kconfig.vkconfig 2006-02-06 22:14:52.000000000 +0300
+++ ./arch/m68knommu/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -644,6 +644,8 @@ source "arch/m68knommu/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/mips/Kconfig.vkconfig 2006-02-06 22:14:52.000000000 +0300
+++ ./arch/mips/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -1811,6 +1811,8 @@ source "arch/mips/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/parisc/Kconfig.vkconfig 2006-02-06 22:15:14.000000000 +0300
+++ ./arch/parisc/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -210,6 +210,8 @@ source "arch/parisc/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/powerpc/Kconfig.vkconfig 2006-02-06 22:14:52.000000000 +0300
+++ ./arch/powerpc/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -975,4 +975,6 @@ config KEYS_COMPAT
   depends on COMPAT && KEYS
   default y
```

```
+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"
--- ./arch/ppc/Kconfig.vkconfig 2006-02-06 22:14:53.000000000 +0300
+++ ./arch/ppc/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -1396,4 +1396,6 @@ source "arch/ppc/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"
--- ./arch/s390/Kconfig.vkconfig 2006-02-06 22:14:53.000000000 +0300
+++ ./arch/s390/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -470,6 +470,8 @@ source "arch/s390/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/sh/Kconfig.vkconfig 2006-02-06 22:14:53.000000000 +0300
+++ ./arch/sh/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -635,6 +635,8 @@ source "arch/sh/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/sh64/Kconfig.vkconfig 2006-02-06 22:14:53.000000000 +0300
+++ ./arch/sh64/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -272,6 +272,8 @@ source "arch/sh64/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/sparc/Kconfig.vkconfig 2006-02-06 22:14:53.000000000 +0300
+++ ./arch/sparc/Kconfig 2006-02-06 23:30:51.000000000 +0300
```

```
@@ -286,6 +286,8 @@ source "arch/sparc/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/sparc64/Kconfig.vkconfig 2006-02-06 22:14:53.000000000 +0300
+++ ./arch/sparc64/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -395,6 +395,8 @@ source "arch/sparc64/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/um/Kconfig.vkconfig 2006-02-06 22:14:53.000000000 +0300
+++ ./arch/um/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -292,6 +292,8 @@ source "fs/Kconfig"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/v850/Kconfig.vkconfig 2006-02-06 22:14:54.000000000 +0300
+++ ./arch/v850/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -318,6 +318,8 @@ source "arch/v850/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/x86_64/Kconfig.vkconfig 2006-02-06 22:14:54.000000000 +0300
+++ ./arch/x86_64/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -604,6 +604,8 @@ source "arch/x86_64/Kconfig.debug"

 source "security/Kconfig"
```

```
+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- ./arch/xtensa/Kconfig.vkconfig 2006-02-06 22:14:54.000000000 +0300
+++ ./arch/xtensa/Kconfig 2006-02-06 23:30:51.000000000 +0300
@@ -247,6 +247,8 @@ source "arch/xtensa/Kconfig.debug"

 source "security/Kconfig"

+source "kernel/Kconfig.container"
+
 source "crypto/Kconfig"

 source "lib/Kconfig"
--- /dev/null  23:22:33.000000000 +0300
+++ ./kernel/Kconfig.container 2006-02-06 23:22:33.000000000 +0300
@@ -0,0 +1,11 @@
+menu "Virtual Containers"
+
+config CONTAINER
+ bool "Virtual Containers support"
+ default n
+ help
+   This option enables support of virtual linux containers,
+   which can be used for creation of virtual environments,
+   Virtual Private Servers, checkpointing, isolation and so on
+
+endmenu
```

Subject: [PATCH 3/4] Virtualization/containers: UID hash
Posted by Kirill Korotaev on Mon, 06 Feb 2006 22:15:04 GMT
View Forum Message <> Reply to Message

This patch virtualizes UID hash, so that processes in container can use
it's own UID set.
Can be done as an option if some virtualization solutions do not require it.

Signed-Off-By: Kirill Korotaev <dev@openvz.org>

Kirill

```
--- ./include/linux/container.h.uids 2006-02-06 23:46:40.000000000 +0300
+++ ./include/linux/container.h 2006-02-07 00:05:33.000000000 +0300
@@ -6,11 +6,14 @@
 #include <asm/atomic.h>
```

```
 struct task_struct;
+struct list_head;

 struct container {
  u32 id;
  struct task_struct *init_task;
  atomic_t refcnt;
+
+ struct list_head *c_uid_hash;
 };

 extern struct container init_container;
--- ./kernel/user.c.uids 2006-02-06 22:15:06.000000000 +0300
+++ ./kernel/user.c 2006-02-06 23:58:06.000000000 +0300
@@ -14,6 +14,7 @@
 #include <linux/bitops.h>
 #include <linux/key.h>
 #include <linux/interrupt.h>
+#include <linux/container.h>

 /*
  * UID task count cache, to get fast user lookup in "alloc_uid"
@@ -24,7 +25,12 @@
 #define UIDHASH_SZ  (1 << UIDHASH_BITS)
 #define UIDHASH_MASK  (UIDHASH_SZ - 1)
 #define __uidhashfn(uid) (((uid >> UIDHASH_BITS) + uid) & UIDHASH_MASK)
+
+#ifdef CONFIG_CONTAINER
+#define uidhashentry(uid) (econtainer()->c_uid_hash + __uidhashfn((uid)))
+#else
 #define uidhashentry(uid) (uidhash_table + __uidhashfn((uid)))
+#endif

 static kmem_cache_t *uid_cachep;
 static struct list_head uidhash_table[UIDHASH_SZ];
@@ -200,6 +206,9 @@ static int __init uid_cache_init(void)

  /* Insert the root user immediately (init already runs as root) */
  spin_lock_irq(&uidhash_lock);
+#ifdef CONFIG_CONTAINER
+ init_container.c_uid_hash = uidhash_table;
+#endif
  uid_hash_insert(&root_user, uidhashentry(0));
  spin_unlock_irq(&uidhash_lock);
```

Subject: [PATCH 4/4] Virtualization/containers: uts name
Posted by Kirill Korotaev on Mon, 06 Feb 2006 22:20:49 GMT
View Forum Message <> Reply to Message

This patch virtualizes uts name.
main changes are done in container.h, uts_name.h,
all other places are just replacement of system_utsname with uts_name.

Signed-Off-By: Kirill Korotaev <dev@openvz.org>

Kirill

```
--- ./arch/alpha/kernel/osf_sys.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/alpha/kernel/osf_sys.c 2006-02-07 01:18:50.000000000 +0300
@@ -402,15 +402,15 @@ osf_utsname(char __user *name)

  down_read(&uts_sem);
  error = -EFAULT;
- if (copy_to_user(name + 0, system_utsname.sysname, 32))
+ if (copy_to_user(name + 0, uts_name.sysname, 32))
    goto out;
- if (copy_to_user(name + 32, system_utsname.nodename, 32))
+ if (copy_to_user(name + 32, uts_name.nodename, 32))
    goto out;
- if (copy_to_user(name + 64, system_utsname.release, 32))
+ if (copy_to_user(name + 64, uts_name.release, 32))
    goto out;
- if (copy_to_user(name + 96, system_utsname.version, 32))
+ if (copy_to_user(name + 96, uts_name.version, 32))
    goto out;
- if (copy_to_user(name + 128, system_utsname.machine, 32))
+ if (copy_to_user(name + 128, uts_name.machine, 32))
    goto out;

  error = 0;
@@ -449,8 +449,8 @@ osf_getdomainname(char __user *name, int

  down_read(&uts_sem);
  for (i = 0; i < len; ++i) {
-  __put_user(system_utsname.domainname[i], name + i);
-  if (system_utsname.domainname[i] == '\0')
+  __put_user(uts_name.domainname[i], name + i);
+  if (uts_name.domainname[i] == '\0')
     break;
  }
  up_read(&uts_sem);
@@ -608,11 +608,11 @@ asmlinkage long
 osf_sysinfo(int command, char __user *buf, long count)
 {
```

```
  static char * sysinfo_table[] = {
-  system_utsname.sysname,
-  system_utsname.nodename,
-  system_utsname.release,
-  system_utsname.version,
-  system_utsname.machine,
+  uts_name.sysname,
+  uts_name.nodename,
+  uts_name.release,
+  uts_name.version,
+  uts_name.machine,
    "alpha", /* instruction set architecture */
    "dummy", /* hardware serial number */
    "dummy", /* hardware manufacturer */
--- ./arch/i386/kernel/sys_i386.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/i386/kernel/sys_i386.c 2006-02-07 01:18:50.000000000 +0300
@@ -217,7 +217,7 @@ asmlinkage int sys_uname(struct old_utsn
  if (!name)
   return -EFAULT;
  down_read(&uts_sem);
- err=copy_to_user(name, &system_utsname, sizeof (*name));
+ err=copy_to_user(name, &uts_name, sizeof (*name));
  up_read(&uts_sem);
  return err?-EFAULT:0;
 }
@@ -233,15 +233,15 @@ asmlinkage int sys_olduname(struct oldol

    down_read(&uts_sem);

- error = __copy_to_user(&name->sysname,&system_utsname.sysname,__OLD_UTS_LEN);
+ error = __copy_to_user(&name->sysname,&uts_name.sysname,__OLD_UTS_LEN);
  error |= __put_user(0,name->sysname+__OLD_UTS_LEN);
- error |= __copy_to_user(&name->nodename,&system_utsname.nodename,__OLD_UTS_LEN);
+ error |= __copy_to_user(&name->nodename,&uts_name.nodename,__OLD_UTS_LEN);
  error |= __put_user(0,name->nodename+__OLD_UTS_LEN);
- error |= __copy_to_user(&name->release,&system_utsname.release,__OLD_UTS_LEN);
+ error |= __copy_to_user(&name->release,&uts_name.release,__OLD_UTS_LEN);
  error |= __put_user(0,name->release+__OLD_UTS_LEN);
- error |= __copy_to_user(&name->version,&system_utsname.version,__OLD_UTS_LEN);
+ error |= __copy_to_user(&name->version,&uts_name.version,__OLD_UTS_LEN);
  error |= __put_user(0,name->version+__OLD_UTS_LEN);
- error |= __copy_to_user(&name->machine,&system_utsname.machine,__OLD_UTS_LEN);
+ error |= __copy_to_user(&name->machine,&uts_name.machine,__OLD_UTS_LEN);
  error |= __put_user(0,name->machine+__OLD_UTS_LEN);

  up_read(&uts_sem);
--- ./arch/m32r/kernel/sys_m32r.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/m32r/kernel/sys_m32r.c 2006-02-07 01:18:50.000000000 +0300
```

```
@@ -199,7 +199,7 @@ asmlinkage int sys_uname(struct old_utsn
  if (!name)
   return -EFAULT;
  down_read(&uts_sem);
- err=copy_to_user(name, &system_utsname, sizeof (*name));
+ err=copy_to_user(name, &uts_name, sizeof (*name));
  up_read(&uts_sem);
  return err?-EFAULT:0;
 }
--- ./arch/mips/kernel/linux32.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/mips/kernel/linux32.c 2006-02-07 01:18:50.000000000 +0300
@@ -1150,7 +1150,7 @@ asmlinkage long sys32_newuname(struct ne
  int ret = 0;

  down_read(&uts_sem);
- if (copy_to_user(name,&system_utsname,sizeof *name))
+ if (copy_to_user(name,&uts_name,sizeof *name))
   ret = -EFAULT;
  up_read(&uts_sem);

--- ./arch/mips/kernel/syscall.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/mips/kernel/syscall.c 2006-02-07 01:18:50.000000000 +0300
@@ -229,7 +229,7 @@ out:
  */
 asmlinkage int sys_uname(struct old_utsname * name)
 {
- if (name && !copy_to_user(name, &system_utsname, sizeof (*name)))
+ if (name && !copy_to_user(name, &uts_name, sizeof (*name)))
   return 0;
  return -EFAULT;
 }
@@ -246,15 +246,15 @@ asmlinkage int sys_olduname(struct oldol
  if (!access_ok(VERIFY_WRITE,name,sizeof(struct oldold_utsname)))
   return -EFAULT;

- error = __copy_to_user(&name->sysname,&system_utsname.sysname,__OLD_UTS_LEN);
+ error = __copy_to_user(&name->sysname,&uts_name.sysname,__OLD_UTS_LEN);
  error -= __put_user(0,name->sysname+__OLD_UTS_LEN);
- error -= __copy_to_user(&name->nodename,&system_utsname.nodename,__OLD_UTS_LEN);
+ error -= __copy_to_user(&name->nodename,&uts_name.nodename,__OLD_UTS_LEN);
  error -= __put_user(0,name->nodename+__OLD_UTS_LEN);
- error -= __copy_to_user(&name->release,&system_utsname.release,__OLD_UTS_LEN);
+ error -= __copy_to_user(&name->release,&uts_name.release,__OLD_UTS_LEN);
  error -= __put_user(0,name->release+__OLD_UTS_LEN);
- error -= __copy_to_user(&name->version,&system_utsname.version,__OLD_UTS_LEN);
+ error -= __copy_to_user(&name->version,&uts_name.version,__OLD_UTS_LEN);
  error -= __put_user(0,name->version+__OLD_UTS_LEN);
- error -= __copy_to_user(&name->machine,&system_utsname.machine,__OLD_UTS_LEN);
```

```
+ error -= __copy_to_user(&name->machine,&uts_name.machine,__OLD_UTS_LEN);
  error = __put_user(0,name->machine+__OLD_UTS_LEN);
  error = error ? -EFAULT : 0;

@@ -290,10 +290,10 @@ asmlinkage int _sys_sysmips(int cmd, lon
     return -EFAULT;

   down_write(&uts_sem);
-  strncpy(system_utsname.nodename, nodename, len);
+  strncpy(uts_name.nodename, nodename, len);
   nodename[__NEW_UTS_LEN] = '\0';
-  strlcpy(system_utsname.nodename, nodename,
-          sizeof(system_utsname.nodename));
+  strlcpy(uts_name.nodename, nodename,
+          sizeof(uts_name.nodename));
   up_write(&uts_sem);
   return 0;
  }
--- ./arch/mips/kernel/sysirix.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/mips/kernel/sysirix.c 2006-02-07 01:18:50.000000000 +0300
@@ -904,7 +904,7 @@ asmlinkage int irix_getdomainname(char _
  down_read(&uts_sem);
  if (len > __NEW_UTS_LEN)
   len = __NEW_UTS_LEN;
- err = copy_to_user(name, system_utsname.domainname, len) ? -EFAULT : 0;
+ err = copy_to_user(name, uts_name.domainname, len) ? -EFAULT : 0;
  up_read(&uts_sem);

  return err;
@@ -1147,11 +1147,11 @@ struct iuname {
 asmlinkage int irix_uname(struct iuname __user *buf)
 {
  down_read(&uts_sem);
- if (copy_from_user(system_utsname.sysname, buf->sysname, 65)
-     || copy_from_user(system_utsname.nodename, buf->nodename, 65)
-     || copy_from_user(system_utsname.release, buf->release, 65)
-     || copy_from_user(system_utsname.version, buf->version, 65)
-     || copy_from_user(system_utsname.machine, buf->machine, 65)) {
+ if (copy_from_user(uts_name.sysname, buf->sysname, 65)
+     || copy_from_user(uts_name.nodename, buf->nodename, 65)
+     || copy_from_user(uts_name.release, buf->release, 65)
+     || copy_from_user(uts_name.version, buf->version, 65)
+     || copy_from_user(uts_name.machine, buf->machine, 65)) {
   return -EFAULT;
  }
  up_read(&uts_sem);
--- ./arch/parisc/hpux/sys_hpux.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/parisc/hpux/sys_hpux.c 2006-02-07 01:18:50.000000000 +0300
```

```
@@ -266,15 +266,15 @@ static int hpux_uname(struct hpux_utsnam

  down_read(&uts_sem);

- error = __copy_to_user(&name->sysname,&system_utsname.sysname,HPUX_UTSLEN-1);
+ error = __copy_to_user(&name->sysname,&uts_name.sysname,HPUX_UTSLEN-1);
  error |= __put_user(0,name->sysname+HPUX_UTSLEN-1);
- error |= __copy_to_user(&name->nodename,&system_utsname.nodename,HPUX_UTSLEN-1);
+ error |= __copy_to_user(&name->nodename,&uts_name.nodename,HPUX_UTSLEN-1);
  error |= __put_user(0,name->nodename+HPUX_UTSLEN-1);
- error |= __copy_to_user(&name->release,&system_utsname.release,HPUX_UTSLEN-1);
+ error |= __copy_to_user(&name->release,&uts_name.release,HPUX_UTSLEN-1);
  error |= __put_user(0,name->release+HPUX_UTSLEN-1);
- error |= __copy_to_user(&name->version,&system_utsname.version,HPUX_UTSLEN-1);
+ error |= __copy_to_user(&name->version,&uts_name.version,HPUX_UTSLEN-1);
  error |= __put_user(0,name->version+HPUX_UTSLEN-1);
- error |= __copy_to_user(&name->machine,&system_utsname.machine,HPUX_UTSLEN-1);
+ error |= __copy_to_user(&name->machine,&uts_name.machine,HPUX_UTSLEN-1);
  error |= __put_user(0,name->machine+HPUX_UTSLEN-1);

  up_read(&uts_sem);
@@ -373,8 +373,8 @@ int hpux_utssys(char *ubuf, int n, int t
   /*  TODO:  print a warning about using this?  */
   down_write(&uts_sem);
   error = -EFAULT;
-  if (!copy_from_user(system_utsname.sysname, ubuf, len)) {
-   system_utsname.sysname[len] = 0;
+  if (!copy_from_user(uts_name.sysname, ubuf, len)) {
+   uts_name.sysname[len] = 0;
    error = 0;
   }
   up_write(&uts_sem);
@@ -400,8 +400,8 @@ int hpux_utssys(char *ubuf, int n, int t
   /*  TODO:  print a warning about this?  */
   down_write(&uts_sem);
   error = -EFAULT;
-  if (!copy_from_user(system_utsname.release, ubuf, len)) {
-   system_utsname.release[len] = 0;
+  if (!copy_from_user(uts_name.release, ubuf, len)) {
+   uts_name.release[len] = 0;
    error = 0;
   }
   up_write(&uts_sem);
@@ -422,13 +422,13 @@ int hpux_getdomainname(char *name, int l

   down_read(&uts_sem);

- nlen = strlen(system_utsname.domainname) + 1;
```

```
+ nlen = strlen(uts_name.domainname) + 1;

  if (nlen < len)
   len = nlen;
  if(len > __NEW_UTS_LEN)
   goto done;
- if(copy_to_user(name, system_utsname.domainname, len))
+ if(copy_to_user(name, uts_name.domainname, len))
   goto done;
  err = 0;
 done:
--- ./arch/powerpc/kernel/syscalls.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/powerpc/kernel/syscalls.c 2006-02-07 01:18:50.000000000 +0300
@@ -259,7 +259,7 @@ long ppc_newuname(struct new_utsname __u
  int err = 0;

  down_read(&uts_sem);
- if (copy_to_user(name, &system_utsname, sizeof(*name)))
+ if (copy_to_user(name, &uts_name, sizeof(*name)))
   err = -EFAULT;
  up_read(&uts_sem);
  if (!err)
@@ -272,7 +272,7 @@ int sys_uname(struct old_utsname __user
  int err = 0;

  down_read(&uts_sem);
- if (copy_to_user(name, &system_utsname, sizeof(*name)))
+ if (copy_to_user(name, &uts_name, sizeof(*name)))
   err = -EFAULT;
  up_read(&uts_sem);
  if (!err)
@@ -288,19 +288,19 @@ int sys_olduname(struct oldold_utsname _
   return -EFAULT;

  down_read(&uts_sem);
- error = __copy_to_user(&name->sysname, &system_utsname.sysname,
+ error = __copy_to_user(&name->sysname, &uts_name.sysname,
          __OLD_UTS_LEN);
  error |= __put_user(0, name->sysname + __OLD_UTS_LEN);
- error |= __copy_to_user(&name->nodename, &system_utsname.nodename,
+ error |= __copy_to_user(&name->nodename, &uts_name.nodename,
     __OLD_UTS_LEN);
  error |= __put_user(0, name->nodename + __OLD_UTS_LEN);
- error |= __copy_to_user(&name->release, &system_utsname.release,
+ error |= __copy_to_user(&name->release, &uts_name.release,
     __OLD_UTS_LEN);
  error |= __put_user(0, name->release + __OLD_UTS_LEN);
- error |= __copy_to_user(&name->version, &system_utsname.version,
```

```
+ error |= __copy_to_user(&name->version, &uts_name.version,
     __OLD_UTS_LEN);
  error |= __put_user(0, name->version + __OLD_UTS_LEN);
- error |= __copy_to_user(&name->machine, &system_utsname.machine,
+ error |= __copy_to_user(&name->machine, &uts_name.machine,
     __OLD_UTS_LEN);
  error |= override_machine(name->machine);
  up_read(&uts_sem);
--- ./arch/sh/kernel/setup.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/sh/kernel/setup.c 2006-02-07 01:18:50.000000000 +0300
@@ -485,7 +485,7 @@ static int show_cpuinfo(struct seq_file
   seq_printf(m, "machine\t\t: %s\n", get_system_type());

  seq_printf(m, "processor\t: %d\n", cpu);
- seq_printf(m, "cpu family\t: %s\n", system_utsname.machine);
+ seq_printf(m, "cpu family\t: %s\n", uts_name.machine);
  seq_printf(m, "cpu type\t: %s\n", get_cpu_subtype());

  show_cpuflags(m);
--- ./arch/sh/kernel/sys_sh.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/sh/kernel/sys_sh.c 2006-02-07 01:18:50.000000000 +0300
@@ -267,7 +267,7 @@ asmlinkage int sys_uname(struct old_utsn
  if (!name)
   return -EFAULT;
  down_read(&uts_sem);
- err=copy_to_user(name, &system_utsname, sizeof (*name));
+ err=copy_to_user(name, &uts_name, sizeof (*name));
  up_read(&uts_sem);
  return err?-EFAULT:0;
 }
--- ./arch/sh64/kernel/sys_sh64.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/sh64/kernel/sys_sh64.c 2006-02-07 01:18:50.000000000 +0300
@@ -279,7 +279,7 @@ asmlinkage int sys_uname(struct old_utsn
  if (!name)
   return -EFAULT;
  down_read(&uts_sem);
- err=copy_to_user(name, &system_utsname, sizeof (*name));
+ err=copy_to_user(name, &uts_name, sizeof (*name));
  up_read(&uts_sem);
  return err?-EFAULT:0;
 }
--- ./arch/sparc/kernel/sys_sparc.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/sparc/kernel/sys_sparc.c 2006-02-07 01:18:50.000000000 +0300
@@ -470,13 +470,13 @@ asmlinkage int sys_getdomainname(char __

   down_read(&uts_sem);

- nlen = strlen(system_utsname.domainname) + 1;
```

```
+ nlen = strlen(uts_name.domainname) + 1;

  if (nlen < len)
   len = nlen;
  if (len > __NEW_UTS_LEN)
   goto done;
- if (copy_to_user(name, system_utsname.domainname, len))
+ if (copy_to_user(name, uts_name.domainname, len))
   goto done;
  err = 0;
 done:
--- ./arch/sparc/kernel/sys_sunos.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/sparc/kernel/sys_sunos.c 2006-02-07 01:18:50.000000000 +0300
@@ -483,13 +483,13 @@ asmlinkage int sunos_uname(struct sunos_
 {
  int ret;
  down_read(&uts_sem);
- ret = copy_to_user(&name->sname[0], &system_utsname.sysname[0], sizeof(name->sname) -
1);
+ ret = copy_to_user(&name->sname[0], &uts_name.sysname[0], sizeof(name->sname) - 1);
  if (!ret) {
-  ret |= __copy_to_user(&name->nname[0], &system_utsname.nodename[0],
sizeof(name->nname) - 1);
+  ret |= __copy_to_user(&name->nname[0], &uts_name.nodename[0], sizeof(name->nname) -
1);
   ret |= __put_user('\0', &name->nname[8]);
-  ret |= __copy_to_user(&name->rel[0], &system_utsname.release[0], sizeof(name->rel) - 1);
-  ret |= __copy_to_user(&name->ver[0], &system_utsname.version[0], sizeof(name->ver) - 1);
-  ret |= __copy_to_user(&name->mach[0], &system_utsname.machine[0], sizeof(name->mach) -
1);
+  ret |= __copy_to_user(&name->rel[0], &uts_name.release[0], sizeof(name->rel) - 1);
+  ret |= __copy_to_user(&name->ver[0], &uts_name.version[0], sizeof(name->ver) - 1);
+  ret |= __copy_to_user(&name->mach[0], &uts_name.machine[0], sizeof(name->mach) - 1);
  }
  up_read(&uts_sem);
  return ret ? -EFAULT : 0;
--- ./arch/sparc64/kernel/sys_sparc.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/sparc64/kernel/sys_sparc.c 2006-02-07 01:18:50.000000000 +0300
@@ -476,13 +476,13 @@ asmlinkage long sys_getdomainname(char _

   down_read(&uts_sem);

- nlen = strlen(system_utsname.domainname) + 1;
+ nlen = strlen(uts_name.domainname) + 1;

        if (nlen < len)
                len = nlen;
  if (len > __NEW_UTS_LEN)
```

```
   goto done;
- if (copy_to_user(name, system_utsname.domainname, len))
+ if (copy_to_user(name, uts_name.domainname, len))
   goto done;
  err = 0;
 done:
--- ./arch/sparc64/kernel/sys_sunos32.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/sparc64/kernel/sys_sunos32.c 2006-02-07 01:18:50.000000000 +0300
@@ -439,16 +439,16 @@ asmlinkage int sunos_uname(struct sunos_
  int ret;

  down_read(&uts_sem);
- ret = copy_to_user(&name->sname[0], &system_utsname.sysname[0],
+ ret = copy_to_user(&name->sname[0], &uts_name.sysname[0],
       sizeof(name->sname) - 1);
- ret |= copy_to_user(&name->nname[0], &system_utsname.nodename[0],
+ ret |= copy_to_user(&name->nname[0], &uts_name.nodename[0],
        sizeof(name->nname) - 1);
  ret |= put_user('\0', &name->nname[8]);
- ret |= copy_to_user(&name->rel[0], &system_utsname.release[0],
+ ret |= copy_to_user(&name->rel[0], &uts_name.release[0],
        sizeof(name->rel) - 1);
- ret |= copy_to_user(&name->ver[0], &system_utsname.version[0],
+ ret |= copy_to_user(&name->ver[0], &uts_name.version[0],
        sizeof(name->ver) - 1);
- ret |= copy_to_user(&name->mach[0], &system_utsname.machine[0],
+ ret |= copy_to_user(&name->mach[0], &uts_name.machine[0],
        sizeof(name->mach) - 1);
  up_read(&uts_sem);
  return (ret ? -EFAULT : 0);
--- ./arch/sparc64/solaris/misc.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/sparc64/solaris/misc.c 2006-02-07 01:18:50.000000000 +0300
@@ -239,7 +239,7 @@ asmlinkage int solaris_utssys(u32 buf, u
   /* Let's cheat */
   err  = set_utsfield(v->sysname, "SunOS", 1, 0);
   down_read(&uts_sem);
-  err |= set_utsfield(v->nodename, system_utsname.nodename,
+  err |= set_utsfield(v->nodename, uts_name.nodename,
         1, 1);
   up_read(&uts_sem);
   err |= set_utsfield(v->release, "2.6", 0, 0);
@@ -263,7 +263,7 @@ asmlinkage int solaris_utsname(u32 buf)
  /* Why should we not lie a bit? */
  down_read(&uts_sem);
  err  = set_utsfield(v->sysname, "SunOS", 0, 0);
- err |= set_utsfield(v->nodename, system_utsname.nodename, 1, 1);
+ err |= set_utsfield(v->nodename, uts_name.nodename, 1, 1);
  err |= set_utsfield(v->release, "5.6", 0, 0);
```

```
  err |= set_utsfield(v->version, "Generic", 0, 0);
  err |= set_utsfield(v->machine, machine(), 0, 0);
@@ -295,7 +295,7 @@ asmlinkage int solaris_sysinfo(int cmd,
  case SI_HOSTNAME:
   r = buffer + 256;
   down_read(&uts_sem);
-  for (p = system_utsname.nodename, q = buffer;
+  for (p = uts_name.nodename, q = buffer;
        q < r && *p && *p != '.'; *q++ = *p++);
   up_read(&uts_sem);
   *q = 0;
--- ./arch/um/kernel/syscall_kern.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/um/kernel/syscall_kern.c 2006-02-07 01:18:50.000000000 +0300
@@ -110,7 +110,7 @@ long sys_uname(struct old_utsname * name
  if (!name)
   return -EFAULT;
  down_read(&uts_sem);
- err=copy_to_user(name, &system_utsname, sizeof (*name));
+ err=copy_to_user(name, &uts_name, sizeof (*name));
  up_read(&uts_sem);
  return err?-EFAULT:0;
 }
@@ -126,19 +126,19 @@ long sys_olduname(struct oldold_utsname

    down_read(&uts_sem);

- error = __copy_to_user(&name->sysname,&system_utsname.sysname,
+ error = __copy_to_user(&name->sysname,&uts_name.sysname,
          __OLD_UTS_LEN);
  error |= __put_user(0,name->sysname+__OLD_UTS_LEN);
- error |= __copy_to_user(&name->nodename,&system_utsname.nodename,
+ error |= __copy_to_user(&name->nodename,&uts_name.nodename,
     __OLD_UTS_LEN);
  error |= __put_user(0,name->nodename+__OLD_UTS_LEN);
- error |= __copy_to_user(&name->release,&system_utsname.release,
+ error |= __copy_to_user(&name->release,&uts_name.release,
     __OLD_UTS_LEN);
  error |= __put_user(0,name->release+__OLD_UTS_LEN);
- error |= __copy_to_user(&name->version,&system_utsname.version,
+ error |= __copy_to_user(&name->version,&uts_name.version,
     __OLD_UTS_LEN);
  error |= __put_user(0,name->version+__OLD_UTS_LEN);
- error |= __copy_to_user(&name->machine,&system_utsname.machine,
+ error |= __copy_to_user(&name->machine,&uts_name.machine,
     __OLD_UTS_LEN);
  error |= __put_user(0,name->machine+__OLD_UTS_LEN);

--- ./arch/um/sys-x86_64/syscalls.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
```

```
+++ ./arch/um/sys-x86_64/syscalls.c 2006-02-07 01:18:50.000000000 +0300
@@ -21,7 +21,7 @@ asmlinkage long sys_uname64(struct new_u
 {
  int err;
  down_read(&uts_sem);
- err = copy_to_user(name, &system_utsname, sizeof (*name));
+ err = copy_to_user(name, &uts_name, sizeof (*name));
  up_read(&uts_sem);
  if (personality(current->personality) == PER_LINUX32)
   err |= copy_to_user(&name->machine, "i686", 5);
--- ./arch/x86_64/ia32/sys_ia32.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/x86_64/ia32/sys_ia32.c 2006-02-07 01:18:50.000000000 +0300
@@ -868,13 +868,13 @@ asmlinkage long sys32_olduname(struct ol

    down_read(&uts_sem);

- error = __copy_to_user(&name->sysname,&system_utsname.sysname,__OLD_UTS_LEN);
+ error = __copy_to_user(&name->sysname,&uts_name.sysname,__OLD_UTS_LEN);
   __put_user(0,name->sysname+__OLD_UTS_LEN);
-  __copy_to_user(&name->nodename,&system_utsname.nodename,__OLD_UTS_LEN);
+  __copy_to_user(&name->nodename,&uts_name.nodename,__OLD_UTS_LEN);
   __put_user(0,name->nodename+__OLD_UTS_LEN);
-  __copy_to_user(&name->release,&system_utsname.release,__OLD_UTS_LEN);
+  __copy_to_user(&name->release,&uts_name.release,__OLD_UTS_LEN);
   __put_user(0,name->release+__OLD_UTS_LEN);
-  __copy_to_user(&name->version,&system_utsname.version,__OLD_UTS_LEN);
+  __copy_to_user(&name->version,&uts_name.version,__OLD_UTS_LEN);
   __put_user(0,name->version+__OLD_UTS_LEN);
   {
    char *arch = "x86_64";
@@ -897,7 +897,7 @@ long sys32_uname(struct old_utsname __us
  if (!name)
   return -EFAULT;
  down_read(&uts_sem);
- err=copy_to_user(name, &system_utsname, sizeof (*name));
+ err=copy_to_user(name, &uts_name, sizeof (*name));
  up_read(&uts_sem);
  if (personality(current->personality) == PER_LINUX32)
   err |= copy_to_user(&name->machine, "i686", 5);
--- ./arch/x86_64/kernel/sys_x86_64.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/x86_64/kernel/sys_x86_64.c 2006-02-07 01:18:50.000000000 +0300
@@ -148,7 +148,7 @@ asmlinkage long sys_uname(struct new_uts
 {
  int err;
  down_read(&uts_sem);
- err = copy_to_user(name, &system_utsname, sizeof (*name));
+ err = copy_to_user(name, &uts_name, sizeof (*name));
  up_read(&uts_sem);
```

```
  if (personality(current->personality) == PER_LINUX32)
   err |= copy_to_user(&name->machine, "i686", 5);
--- ./arch/xtensa/kernel/syscalls.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./arch/xtensa/kernel/syscalls.c 2006-02-07 01:18:50.000000000 +0300
@@ -129,7 +129,7 @@ out:

 int sys_uname(struct old_utsname * name)
 {
- if (name && !copy_to_user(name, &system_utsname, sizeof (*name)))
+ if (name && !copy_to_user(name, &uts_name, sizeof (*name)))
   return 0;
  return -EFAULT;
 }
--- ./drivers/usb/core/hcd.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./drivers/usb/core/hcd.c 2006-02-07 01:18:50.000000000 +0300
@@ -317,8 +317,8 @@ static int rh_string (

   // id 3 == vendor description
  } else if (id == 3) {
-  snprintf (buf, sizeof buf, "%s %s %s", system_utsname.sysname,
-   system_utsname.release, hcd->driver->description);
+  snprintf (buf, sizeof buf, "%s %s %s", uts_name.sysname,
+   uts_name.release, hcd->driver->description);

  // unsupported IDs --> "protocol stall"
  } else
--- ./fs/cifs/connect.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./fs/cifs/connect.c 2006-02-07 01:18:50.000000000 +0300
@@ -765,12 +765,12 @@ cifs_parse_mount_options(char *options,
  separator[1] = 0;

  memset(vol->source_rfc1001_name,0x20,15);
- for(i=0;i < strnlen(system_utsname.nodename,15);i++) {
+ for(i=0;i < strnlen(uts_name.nodename,15);i++) {
   /* does not have to be a perfect mapping since the field is
    informational, only used for servers that do not support
    port 445 and it can be overridden at mount time */
    vol->source_rfc1001_name[i] =
-   toupper(system_utsname.nodename[i]);
+   toupper(uts_name.nodename[i]);
  }
  vol->source_rfc1001_name[15] = 0;
  /* null target name indicates to use *SMBSERVR default called name
@@ -2062,7 +2062,7 @@ CIFSSessSetup(unsigned int xid, struct c
       32, nls_codepage);
   bcc_ptr += 2 * bytes_returned;
   bytes_returned =
-      cifs_strtoUCS((__le16 *) bcc_ptr, system_utsname.release,
```

```
+      cifs_strtoUCS((__le16 *) bcc_ptr, uts_name.release,
       32, nls_codepage);
   bcc_ptr += 2 * bytes_returned;
   bcc_ptr += 2;
@@ -2089,8 +2089,8 @@ CIFSSessSetup(unsigned int xid, struct c
   }
   strcpy(bcc_ptr, "Linux version ");
   bcc_ptr += strlen("Linux version ");
-  strcpy(bcc_ptr, system_utsname.release);
-  bcc_ptr += strlen(system_utsname.release) + 1;
+  strcpy(bcc_ptr, uts_name.release);
+  bcc_ptr += strlen(uts_name.release) + 1;
   strcpy(bcc_ptr, CIFS_NETWORK_OPSYS);
   bcc_ptr += strlen(CIFS_NETWORK_OPSYS) + 1;
  }
@@ -2329,7 +2329,7 @@ CIFSSpnegoSessSetup(unsigned int xid, st
       32, nls_codepage);
   bcc_ptr += 2 * bytes_returned;
   bytes_returned =
-      cifs_strtoUCS((__le16 *) bcc_ptr, system_utsname.release, 32,
+      cifs_strtoUCS((__le16 *) bcc_ptr, uts_name.release, 32,
       nls_codepage);
   bcc_ptr += 2 * bytes_returned;
   bcc_ptr += 2;
@@ -2354,8 +2354,8 @@ CIFSSpnegoSessSetup(unsigned int xid, st
   }
   strcpy(bcc_ptr, "Linux version ");
   bcc_ptr += strlen("Linux version ");
-  strcpy(bcc_ptr, system_utsname.release);
-  bcc_ptr += strlen(system_utsname.release) + 1;
+  strcpy(bcc_ptr, uts_name.release);
+  bcc_ptr += strlen(uts_name.release) + 1;
   strcpy(bcc_ptr, CIFS_NETWORK_OPSYS);
   bcc_ptr += strlen(CIFS_NETWORK_OPSYS) + 1;
  }
@@ -2619,7 +2619,7 @@ CIFSNTLMSSPNegotiateSessSetup(unsigned i
       32, nls_codepage);
   bcc_ptr += 2 * bytes_returned;
   bytes_returned =
-      cifs_strtoUCS((__le16 *) bcc_ptr, system_utsname.release, 32,
+      cifs_strtoUCS((__le16 *) bcc_ptr, uts_name.release, 32,
       nls_codepage);
   bcc_ptr += 2 * bytes_returned;
   bcc_ptr += 2; /* null terminate Linux version */
@@ -2636,8 +2636,8 @@ CIFSNTLMSSPNegotiateSessSetup(unsigned i
  } else {  /* ASCII */
   strcpy(bcc_ptr, "Linux version ");
   bcc_ptr += strlen("Linux version ");
```

```
-  strcpy(bcc_ptr, system_utsname.release);
-  bcc_ptr += strlen(system_utsname.release) + 1;
+  strcpy(bcc_ptr, uts_name.release);
+  bcc_ptr += strlen(uts_name.release) + 1;
   strcpy(bcc_ptr, CIFS_NETWORK_OPSYS);
   bcc_ptr += strlen(CIFS_NETWORK_OPSYS) + 1;
   bcc_ptr++; /* empty domain field */
@@ -2998,7 +2998,7 @@ CIFSNTLMSSPAuthSessSetup(unsigned int xi
       32, nls_codepage);
   bcc_ptr += 2 * bytes_returned;
   bytes_returned =
-      cifs_strtoUCS((__le16 *) bcc_ptr, system_utsname.release, 32,
+       cifs_strtoUCS((__le16 *) bcc_ptr, uts_name.release, 32,
       nls_codepage);
   bcc_ptr += 2 * bytes_returned;
   bcc_ptr += 2; /* null term version string */
@@ -3050,8 +3050,8 @@ CIFSNTLMSSPAuthSessSetup(unsigned int xi

   strcpy(bcc_ptr, "Linux version ");
   bcc_ptr += strlen("Linux version ");
-  strcpy(bcc_ptr, system_utsname.release);
-  bcc_ptr += strlen(system_utsname.release) + 1;
+  strcpy(bcc_ptr, uts_name.release);
+  bcc_ptr += strlen(uts_name.release) + 1;
   strcpy(bcc_ptr, CIFS_NETWORK_OPSYS);
   bcc_ptr += strlen(CIFS_NETWORK_OPSYS) + 1;
   bcc_ptr++; /* null domain */
--- ./fs/exec.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./fs/exec.c 2006-02-07 01:18:50.000000000 +0300
@@ -1326,7 +1326,7 @@ static void format_corename(char *corena
    case 'h':
     down_read(&uts_sem);
     rc = snprintf(out_ptr, out_end - out_ptr,
-          "%s", system_utsname.nodename);
+           "%s", uts_name.nodename);
     up_read(&uts_sem);
     if (rc > out_end - out_ptr)
      goto out;
--- ./fs/lockd/clntproc.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./fs/lockd/clntproc.c 2006-02-07 01:18:50.000000000 +0300
@@ -130,10 +130,10 @@ static void nlmclnt_setlockargs(struct n
  nlmclnt_next_cookie(&argp->cookie);
  argp->state   = nsm_local_state;
  memcpy(&lock->fh, NFS_FH(fl->fl_file->f_dentry->d_inode), sizeof(struct nfs_fh));
- lock->caller  = system_utsname.nodename;
+ lock->caller  = uts_name.nodename;
  lock->oh.data = req->a_owner;
  lock->oh.len  = sprintf(req->a_owner, "%d@%s",
```

```
-    current->pid, system_utsname.nodename);
+    current->pid, uts_name.nodename);
  locks_copy_lock(&lock->fl, fl);
 }

@@ -154,7 +154,7 @@ nlmclnt_setgrantargs(struct nlm_rqst *ca
 {
  locks_copy_lock(&call->a_args.lock.fl, &lock->fl);
  memcpy(&call->a_args.lock.fh, &lock->fh, sizeof(call->a_args.lock.fh));
- call->a_args.lock.caller = system_utsname.nodename;
+ call->a_args.lock.caller = uts_name.nodename;
  call->a_args.lock.oh.len = lock->oh.len;

  /* set default data area */
--- ./include/asm-i386/elf.h.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./include/asm-i386/elf.h 2006-02-07 01:18:50.000000000 +0300
@@ -108,7 +108,7 @@ typedef struct user_fxsr_struct elf_fpxr
    For the moment, we have only optimizations for the Intel generations,
    but that could change... */

-#define ELF_PLATFORM  (system_utsname.machine)
+#define ELF_PLATFORM  (uts_name.machine)

 #ifdef __KERNEL__
 #define SET_PERSONALITY(ex, ibcs2) do { } while (0)
--- ./include/linux/container.h.utsnamex 2006-02-07 01:18:40.000000000 +0300
+++ ./include/linux/container.h 2006-02-07 01:18:53.000000000 +0300
@@ -7,6 +7,7 @@

 struct task_struct;
 struct list_head;
+struct new_utsname;

 struct container {
  u32 id;
@@ -14,6 +15,7 @@ struct container {
  atomic_t refcnt;

  struct list_head *c_uid_hash;
+ struct new_utsname *c_uts_name;
 };

 extern struct container init_container;
--- ./include/linux/init_task.h.utsnamex 2006-02-07 01:18:40.000000000 +0300
+++ ./include/linux/init_task.h 2006-02-07 01:18:53.000000000 +0300
@@ -133,6 +133,7 @@ extern struct group_info init_groups;
 }
```

```
 #define INIT_CONTAINER(cont)      \
- .refcnt  = ATOMIC_INIT(1)
+ .refcnt  = ATOMIC_INIT(1)    \
+ .c_uts_name = &system_utsname,

 #endif
--- ./include/linux/utsname.h.utsnamex 2006-02-07 01:18:40.000000000 +0300
+++ ./include/linux/utsname.h 2006-02-07 01:18:53.000000000 +0300
@@ -1,6 +1,8 @@
 #ifndef _LINUX_UTSNAME_H
 #define _LINUX_UTSNAME_H

+#include <linux/container.h>
+
 #define __OLD_UTS_LEN 8

 struct oldold_utsname {
@@ -31,6 +33,11 @@ struct new_utsname {
 };

 extern struct new_utsname system_utsname;
+#ifdef CONFIG_CONTAINER
+#define uts_name (*(econtainer()->c_uts_name))
+#else
+#define uts_name system_utsname
+#endif

 extern struct rw_semaphore uts_sem;
 #endif
--- ./kernel/sys.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./kernel/sys.c 2006-02-07 01:18:50.000000000 +0300
@@ -1518,7 +1518,7 @@ asmlinkage long sys_newuname(struct new_
  int errno = 0;

  down_read(&uts_sem);
- if (copy_to_user(name,&system_utsname,sizeof *name))
+ if (copy_to_user(name,&uts_name,sizeof *name))
   errno = -EFAULT;
  up_read(&uts_sem);
  return errno;
@@ -1536,8 +1536,8 @@ asmlinkage long sys_sethostname(char __u
  down_write(&uts_sem);
  errno = -EFAULT;
  if (!copy_from_user(tmp, name, len)) {
-  memcpy(system_utsname.nodename, tmp, len);
-  system_utsname.nodename[len] = 0;
+  memcpy(uts_name.nodename, tmp, len);
+  uts_name.nodename[len] = 0;
```

```
   errno = 0;
  }
  up_write(&uts_sem);
@@ -1553,11 +1553,11 @@ asmlinkage long sys_gethostname(char __u
  if (len < 0)
   return -EINVAL;
  down_read(&uts_sem);
- i = 1 + strlen(system_utsname.nodename);
+ i = 1 + strlen(uts_name.nodename);
  if (i > len)
   i = len;
  errno = 0;
- if (copy_to_user(name, system_utsname.nodename, i))
+ if (copy_to_user(name, uts_name.nodename, i))
   errno = -EFAULT;
  up_read(&uts_sem);
  return errno;
@@ -1582,8 +1582,8 @@ asmlinkage long sys_setdomainname(char _
  down_write(&uts_sem);
  errno = -EFAULT;
  if (!copy_from_user(tmp, name, len)) {
-  memcpy(system_utsname.domainname, tmp, len);
-  system_utsname.domainname[len] = 0;
+  memcpy(uts_name.domainname, tmp, len);
+  uts_name.domainname[len] = 0;
   errno = 0;
  }
  up_write(&uts_sem);
--- ./net/sunrpc/clnt.c.utsnamex 2006-02-07 01:18:42.000000000 +0300
+++ ./net/sunrpc/clnt.c 2006-02-07 01:18:50.000000000 +0300
@@ -168,10 +168,10 @@ rpc_new_client(struct rpc_xprt *xprt, ch
  }

  /* save the nodename */
- clnt->cl_nodelen = strlen(system_utsname.nodename);
+ clnt->cl_nodelen = strlen(uts_name.nodename);
  if (clnt->cl_nodelen > UNX_MAXNODENAME)
   clnt->cl_nodelen = UNX_MAXNODENAME;
- memcpy(clnt->cl_nodename, system_utsname.nodename, clnt->cl_nodelen);
+ memcpy(clnt->cl_nodename, uts_name.nodename, clnt->cl_nodelen);
  return clnt;

 out_no_auth:
```

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Dave Hansen on Mon, 06 Feb 2006 23:00:58 GMT

On Tue, 2006-02-07 at 00:57 +0300, Kirill Korotaev wrote:
> @@ -1132,6 +1133,7 @@ static task_t *copy_process(unsigned lon
> 	p->ioprio = current->ioprio;
>
> 	SET_LINKS(p);
> +	 (void)get_container(p->container);
> 	if (unlikely(p->ptrace & PT_PTRACED))
> 		__ptrace_link(p, current->parent);

This entire patch looks nice and very straightforward, except for this bit. :) The "(void)" bit isn't usual kernel coding style. You can probably kill it.

BTW, why does get_container() return the container argument? get_task_struct(), for instance is just a do{}while(0) loop, so it doesn't have a return value. Is there some magic later on in your patch set that utilizes this?

One other really minor thing: I usually try to do is keep the ! CONFIG_FOO functions static inlines, just like the full versions. The advantage is that you get some compile-time type checking, even when your CONFIG option is off.

-- Dave

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by ebiederm on Tue, 07 Feb 2006 03:34:33 GMT

Kirill Korotaev <dev@openvz.org> writes:

> Hello,
>
> I tried to take into account all the comments from you guys (thanks a lot for
> them!) and prepared a new version of virtualization patches. I will send only 4
> patches today, just not to overflow everyone and keep it clear/tidy/possible to
> review.
>
> This patch introduces some abstract container kernel structure and a number of
> operations on it.

>
> The important properties of the proposed container implementation:
> - each container has unique ID in the system
What namespace does this ID live in?

> - each process in the kernel can belong to one container only
Reasonable.

> - effective container pointer (econtainer()) is used on the task to avoid
> insertion of additional argument "container" to all functions where it is
> required.

Why is that desirable?

> - kernel compilation with disabled virtualization should result in old good
> linux kernel
A reasonable goal.

Why do we need a container structure to hold pointers to other pointers?

> Patches following this one will be used for virtualization of the kernel
> resources based on this container infrastructure, including those VPID patches I
> sent before. Every virtualized resource can be given separate config option if
> needed (just give me to know if it is desired).
>
> Signed-Off-By: Kirill Korotaev <dev@openvz.org>
>
> Kirill
>
> P.S. I understand that this virtualization spam can be unintersting for some of
> you, just give me to know if you want to be removed from CC.

May I please be added to the CC list.

We are never going to form a consensus if all of the people doing implementations don't talk.

Eric

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Rik van Riel on Tue, 07 Feb 2006 03:40:01 GMT
View Forum Message <> Reply to Message

On Mon, 6 Feb 2006, Eric W. Biederman wrote:

> We are never going to form a consensus if all of the people doing
> implementations don't talk.

Speaking of which - it would be interesting to get Kirill's
comments on Eric's patchset ;)

Once we know what's good and bad about both patchsets, we'll
be a lot closer to knowing what exactly should go upstream.

--
All Rights Reversed

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Sam Vilain on Tue, 07 Feb 2006 06:30:34 GMT
View Forum Message <> Reply to Message

Rik van Riel wrote:
> On Mon, 6 Feb 2006, Eric W. Biederman wrote:
>
>
>>We are never going to form a consensus if all of the people doing
>>implementations don't talk.
>
>
> Speaking of which - it would be interesting to get Kirill's
> comments on Eric's patchset ;)
>
> Once we know what's good and bad about both patchsets, we'll
> be a lot closer to knowing what exactly should go upstream.

Let's compare approaches of patchsets before the patchsets themselves.

It seems to be, should we:

A) make a general form of virtualising PIDs, and hope this assists
later virtualisation efforts (Eric's patch)

B) make a general form of containers/jails/vservers/vpses, and layer
PID virtualisation on top of it somewhere (as in openvz, vserver)

I can't think of any real use cases where you would specifically want A)
without B).

Also, the problem space in B) is now very well explored. To start with
A) would mean to throw away 4+ years of experience at this approach
(just counting vserver and variants - not FreeBSD Jail, etc). Trying to
re-base B) atop a massive refactoring and new patch like A) would incur
a lot of work; however fitting it into B) is natural and solved
conceptually and in practice, with the only drawback I see being that
the use cases mentioned above wouldn't suffer from the side-effects of
B).

Perhaps I'm wrong there, but that's my gut feeling.

Sam.

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by dev on Tue, 07 Feb 2006 11:49:18 GMT
View Forum Message <> Reply to Message

>>> We are never going to form a consensus if all of the people doing
>>> implementations don't talk.
>>
>> Speaking of which - it would be interesting to get Kirill's
>> comments on Eric's patchset ;)
I'll do comment.

>> Once we know what's good and bad about both patchsets, we'll
>> be a lot closer to knowing what exactly should go upstream.
I'm starting to think that nothing in upstream can be better for all of
us :)

> Let's compare approaches of patchsets before the patchsets themselves.
> It seems to be, should we:
>
> A) make a general form of virtualising PIDs, and hope this assists
> later virtualisation efforts (Eric's patch)
>
> B) make a general form of containers/jails/vservers/vpses, and layer
> PID virtualisation on top of it somewhere (as in openvz, vserver)
 >
> I can't think of any real use cases where you would specifically want A)
> without B).
Exactly! All these patches for A) look weird for me without containers
itself. A try to make half-solution which is bad.

> Also, the problem space in B) is now very well explored. To start with
> A) would mean to throw away 4+ years of experience at this approach
> (just counting vserver and variants - not FreeBSD Jail, etc). Trying to
> re-base B) atop a massive refactoring and new patch like A) would incur
> a lot of work; however fitting it into B) is natural and solved
> conceptually and in practice, with the only drawback I see being that
> the use cases mentioned above wouldn't suffer from the side-effects of
> B).
Have you saw my patches?
This is B) :) This is what we should start with IMHO.
Having a containers and isolation all these talks about A) will be much
more precise.

Kirill

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by dev on Tue, 07 Feb 2006 12:12:43 GMT
View Forum Message <> Reply to Message

>>The important properties of the proposed container implementation:
>>- each container has unique ID in the system
> What namespace does this ID live in?
global namespace. can be virtualized later.
can be optional.
But the idea is simple. Eventually you will need some management tools
anyway. And they should be able to refer to containers.

>>- each process in the kernel can belong to one container only
> Reasonable.
>
>
>>- effective container pointer (econtainer()) is used on the task to avoid
>>insertion of additional argument "container" to all functions where it is
>>required.
> Why is that desirable?
It was discussed with Linus and the reason is provided in this text
actually.
There are 2 ways:
- to add additional argument "container" to all the functions where it
is required.
Drawbacks: a) lot's of changes, b) compilation without virtualization is
not the same. c) increased stakc usage
- to add effective container pointer on the task. i.e. context which
kernel should be in when works with virtualized resources.
Drawbacks: a) there are some places where you need to change effective
container context explicitly such as TCP/IP.

>>- kernel compilation with disabled virtualization should result in old good
>>linux kernel
>
> A reasonable goal.
>
> Why do we need a container structure to hold pointers to other pointers?
can't catch what you mean :) is it prohibited somehow? :))))

> May I please be added to the CC list.
> We are never going to form a consensus if all of the people doing implementations don't
> talk.
To make a consensus people need to make mutual concessions... Otherwise
these talks are useless.

Kirill

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by dev on Tue, 07 Feb 2006 12:22:18 GMT
View Forum Message <> Reply to Message

>>@@ -1132,6 +1133,7 @@ static task_t *copy_process(unsigned lon
>> p->ioprio = current->ioprio;
>>
>> SET_LINKS(p);
>>+ (void)get_container(p->container);
>> if (unlikely(p->ptrace & PT_PTRACED))
>> __ptrace_link(p, current->parent);
>
>
> This entire patch looks nice and very straightforward, except for this
> bit. :) The "(void)" bit isn't usual kernel coding style. You can
> probably kill it.
it is to avoid warning message the value has no effect.

> BTW, why does get_container() return the container argument?
> get_task_struct(), for instance is just a do{}while(0) loop, so it
> doesn't have a return value. Is there some magic later on in your patch
> set that utilizes this?
ok, I will remake it without a return value. not a real problem at all.

> One other really minor thing: I usually try to do is keep the !
> CONFIG_FOO functions static inlines, just like the full versions. The
> advantage is that you get some compile-time type checking, even when
> your CONFIG option is off.
it is not always appropriate :(I try to follow this as well :)

Kirill

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by ebiederm on Tue, 07 Feb 2006 14:06:50 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

>>>The important properties of the proposed container implementation:
>>>- each container has unique ID in the system
>> What namespace does this ID live in?
> global namespace. can be virtualized later.
> can be optional.

So a totally new kind of number in the kernel, that doesn't fall
in any of the traditional namespaces. That is something I would
like to avoid.

> But the idea is simple. Eventually you will need some management tools
> anyway. And they should be able to refer to containers.

In my implementation I refer to things indirectly via the task
in it since I have an appropriate pid. As does the current
kernel code for the filesystem mount namespace.

So far except a tool to create my things I don't need special
management tools the traditional unix ones suffice as I am
simply extending the traditional unix model.

The best counter argument I have is that filesystem namespaces
have already been implemented and we have not needed an additional id
in the kernel.

>>>- each process in the kernel can belong to one container only
>> Reasonable.
>>
>>>- effective container pointer (econtainer()) is used on the task to avoid
>>>insertion of additional argument "container" to all functions where it is
>>>required.
>> Why is that desirable?

Ok. I can finally see the effective thing. Writing an argument onto
the task structure instead of onto the stack seems weird to me.

> It was discussed with Linus and the reason is provided in this text actually.
> There are 2 ways:
> - to add additional argument "container" to all the functions where it is
> required.
> Drawbacks:

> a) lot's of changes,

I believe most of those changes make other places in the kernel where
misuse happens easier to find. Turning places that need changes into
compile errors seems a virtue to me. I do know there are places that
store identifiers that might be used in a different context later on.
Finding all of those places seems more important than generating compile
errors for a few simple cases.

> b) compilation without virtualization is not the same.

Which is a reasonable consideration. But I expect most common kernels
will ship with virtualization enabled so that is the important case to
get correct.

> c) increased stakc usage

I think we are talking a few extra pointers on the stack. Our call nesting depth is not that great. So I would be surprised if the additional stack usage would be significant. I expect the gains from the locality of having everything in the same cache line are greater, than the gains from the reduced stack usage.

> - to add effective container pointer on the task. i.e. context which kernel
> should be in when works with virtualized resources.
> Drawbacks: a) there are some places where you need to change effective container
> context explicitly such as TCP/IP.

Yes.

Another is that it is easier to be aware and to think about what context you are working in if it is explicit. I expect other kernel developers are more likely to get it right if what is going is not hidden from them.

>>>- kernel compilation with disabled virtualization should result in old good
>>>linux kernel
>> A reasonable goal.
>> Why do we need a container structure to hold pointers to other pointers?
> can't catch what you mean :) is it prohibited somehow? :))))

No I just think an additional structure is unnecessary. Possibly it is a reasonable thing to do.

>> May I please be added to the CC list.
>> We are never going to form a consensus if all of the people doing
> implementations don't
>> talk.

> To make a consensus people need to make mutual concessions... Otherwise these
> talks are useless.

Consensus does not require concessions. We are after technical excellence after all.

Yes people have to be willing to bend to work together to find the best solution. I am willing. At the same time I am not easy to convince that other solutions are necessarily better. To understand which is better we need to understand why each of us made the decisions we have made or hold the opinion we hold.

Once our reasons for doing things are clear and we can agree to a common set of goals, picking the best solution should not be too difficult.

My problem in previous talks is that I don't think you have seen where I have
been coming from. Which is why I shut up earlier and posted code.

Eric

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by ebiederm on Tue, 07 Feb 2006 14:31:33 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

>>>> We are never going to form a consensus if all of the people doing
>>>> implementations don't talk.
>>>
>>> Speaking of which - it would be interesting to get Kirill's
>>> comments on Eric's patchset ;)
> I'll do comment.

Thank you I will look forward to your comments.

>>> Once we know what's good and bad about both patchsets, we'll
>>> be a lot closer to knowing what exactly should go upstream.
> I'm starting to think that nothing in upstream can be better for all of us :)

In a thread voicing the concerns for maintaining out of tree patches
that is a natural concern.

>> Let's compare approaches of patchsets before the patchsets themselves.
>> It seems to be, should we:
>> A) make a general form of virtualising PIDs, and hope this assists
>>   later virtualisation efforts (Eric's patch)
>> B) make a general form of containers/jails/vservers/vpses, and layer
>>   PID virtualisation on top of it somewhere (as in openvz, vserver)
> >
>> I can't think of any real use cases where you would specifically want A)
>> without B).
> Exactly! All these patches for A) look weird for me without containers itself. A
> try to make half-solution which is bad.

I am willing to contend that my approach also leads to a complete solution.
In fact I believe my network virtualization has actually gone much farther
than yours. Although I admit there is still some work to do before
the code is in shape to be merged.

You notice in the kernel there is also not a struct process?

To me having a container structure while an obvious approach to the problem
seems to add unnecessary policy to the kernel. Lumping together the
implementation of multiple instances of different namespaces in a way
that the implementation does not require.

Eric

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Rik van Riel on Tue, 07 Feb 2006 14:52:17 GMT
View Forum Message <> Reply to Message

On Tue, 7 Feb 2006, Eric W. Biederman wrote:

> Yes people have to be willing to bend to work together to find the
> best solution. I am willing. At the same time I am not easy to
> convince that other solutions are necessarily better.

It's not about "better", it's about different requirements.

--
All Rights Reversed

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by ebiederm on Tue, 07 Feb 2006 15:13:28 GMT
View Forum Message <> Reply to Message

Rik van Riel <riel@redhat.com> writes:

> On Tue, 7 Feb 2006, Eric W. Biederman wrote:
>
>> Yes people have to be willing to bend to work together to find the
>> best solution. I am willing. At the same time I am not easy to
>> convince that other solutions are necessarily better.
>
> It's not about "better", it's about different requirements.

In part but I think it is possible to largely agree on what the requirements
for the kernel are, by seeing what the requirements of the different
users are. That is part of the discussion.

Once those requirements are agreed upon it is about the best technical
solution.

Maintainability is one of the more important requirements is it not?

Eric

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by ebiederm on Tue, 07 Feb 2006 15:42:12 GMT
View Forum Message <> Reply to Message

Sam Vilain <sam@vilain.net> writes:

> Rik van Riel wrote:
>> On Mon, 6 Feb 2006, Eric W. Biederman wrote:
>>
>>> We are never going to form a consensus if all of the people doing
>>> implementations don't talk.
>> Speaking of which - it would be interesting to get Kirill's
>> comments on Eric's patchset ;)
>> Once we know what's good and bad about both patchsets, we'll
>> be a lot closer to knowing what exactly should go upstream.
>
> Let's compare approaches of patchsets before the patchsets themselves.
>
> It seems to be, should we:
>
> A) make a general form of virtualising PIDs, and hope this assists
> later virtualisation efforts (Eric's patch)
>
> B) make a general form of containers/jails/vservers/vpses, and layer
> PID virtualisation on top of it somewhere (as in openvz, vserver)
>
> I can't think of any real use cases where you would specifically want A)
> without B).

You misrepresent my approach.

First there is a huge commonality in the code bases between the
different implementations and I have already gotten preliminary
acceptance from the vserver developers, that my approach is sane. The
major difference is what user interface does the kernel export,
and I posted my user interface.

What user interface to export is a debate worth having.

For a lot of things getting the details just so is very important
to long term maintainability and it is not my impression that anyone
has done that yet.

Second I am not trying to just implement a form of virtualizing PIDs. Heck I don't intend to virtualize anything. The kernel has already virtualized everything I require. I want to implement multiple instances of the current kernel global namespaces. All I want is to be able to use the same name twice in user space and not have a conflict.

Beyond getting multiple instance of all of the kernel namespaces (which is the hard requirement for migration) my approach is to see what is needed for projects like vserver and vps and see how their needs can be met as well.

I disagree with a struct container simply because I do not see what value it happens to bring to the table. I have yet to see a problem that it solves that I have not solved yet.

In addition I depart from vserver and other implementations in another regard. It is my impression a lot of their work has been done so those projects are maintainable outside of the kernel, which makes sense as that is where those code bases live. But I don't think that gives the best solution for an in kernel implementation, which is what we are implementing.

So far I have succeeded in communicating with both the IBM and vserver developers. Hopefully I can do the same with Kirill Korotaev and the OpenVz team. I think my implementation stands up to criticism. But expect surprises in the way I solve a number of problems.

I suspect I will find similar surprises in the OpenVz code.

Time to do some more research I guess.

Eric

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by dev on Tue, 07 Feb 2006 16:16:36 GMT
View Forum Message <> Reply to Message

>>I can't think of any real use cases where you would specifically want A)
>>without B).

> You misrepresent my approach.
[...]

> Second I am not trying to just implement a form of virtualizing PIDs.
> Heck I don't intend to virtualize anything. The kernel has already
> virtualized everything I require. I want to implement multiple
> instances of the current kernel global namespaces. All I want is
> to be able to use the same name twice in user space and not have
> a conflict.
if you want not virtualize anything, what is this discussion about? :)
can you provide an URL to your sources? you makes lot's of statements
about that your network virtualization solution is better/more complete,
so I'd like to see your solution in whole rather than only words.
Probably this will help.

> I disagree with a struct container simply because I do not see what
> value it happens to bring to the table. I have yet to see a problem
> that it solves that I have not solved yet.
again, source would help to understand your solution and problem you
solved and not solved yet.

> In addition I depart from vserver and other implementations in another
> regard. It is my impression a lot of their work has been done so
> those projects are maintainable outside of the kernel, which makes
> sense as that is where those code bases live. But I don't think that
> gives the best solution for an in kernel implementation, which is
> what we are implementing.
These soltuions are in kernel implementations actually.

Kirill

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Hubertus Franke on Tue, 07 Feb 2006 16:57:04 GMT
View Forum Message <> Reply to Message

Sam Vilain wrote:
> Rik van Riel wrote:
>
>> On Mon, 6 Feb 2006, Eric W. Biederman wrote:
>>
>>
>>> We are never going to form a consensus if all of the people doing
>>> implementations don't talk.
>>
>>

>> Speaking of which - it would be interesting to get Kirill's
>> comments on Eric's patchset ;)
>>
>> Once we know what's good and bad about both patchsets, we'll
>> be a lot closer to knowing what exactly should go upstream.
>
>
> Let's compare approaches of patchsets before the patchsets themselves.
>
> It seems to be, should we:
>
> A) make a general form of virtualising PIDs, and hope this assists
> later virtualisation efforts (Eric's patch)
>
> B) make a general form of containers/jails/vservers/vpses, and layer
> PID virtualisation on top of it somewhere (as in openvz, vserver)
>
> I can't think of any real use cases where you would specifically want A)
> without B).
>
> Also, the problem space in B) is now very well explored. To start with
> A) would mean to throw away 4+ years of experience at this approach
> (just counting vserver and variants - not FreeBSD Jail, etc). Trying to
> re-base B) atop a massive refactoring and new patch like A) would incur
> a lot of work; however fitting it into B) is natural and solved
> conceptually and in practice, with the only drawback I see being that
> the use cases mentioned above wouldn't suffer from the side-effects of
> B).
>
Sam, that is a bit far fetched. I looked and experienced myself with both
approaches and there is a lot of functional overlap, with both of them
having advantages and disadvantages.
What Eric provides is an alternative to the PID virtualization part of openvz.
Indeed it is a pid isolation more then anything else (with some dealing at
the boundary condition).
I personally don't see much problem in replacing the pid virtualization of
openvz with that of pidspaces.
So the correct thing to do here is as RvR points out, simple discuss the
merits and drawbacks of each PID approach for now and settle on one and
move on....

Here are my two cents on this.

The pid-namespace (pspace) provides an approach of fully separate
the allocation and maintenance of the pids and treating the <pspace,pid>
tuple as an entity to uniquely identify a task and vice versa.
As a result the logic of lookup can be pushed down the find_task_by_pid()
lookup. There are specific cases where the init_task of a container or

pspace needs to be checked to ensure that signals/waits and alike are properly
handled across pspace boundaries. I think this is an intuitive and clean way.
It also completely avoids the problem of having to think about all the locations
at the user/kernel boundary where a vpid/pid conversion needs to take place.
It also avoids the problems that logically vpids and pids are different types and
therefore it would have been good to have type checking automatically identify
problem spots.
On the negative side, it does require to maintain a pidmap per pidspace.

The vpid approach has the drawbacks of having to identify the conversion spots
of all vpid vs. pid semantics. On the otherhand it does take advantage
of the fact that no virtualization has to take place until a "container"
has been migrated, thus rendering most of the vpid<->pid calls to be
noops.

What I like about the pspace approach is that it explicitely defines in the code
when I am using a different pspace for the lookup. That is kind of hidden in
the vpid/pid approach.

The container is just an umbrella object that ties every "virtualized" subsystem
together.

So, what do other folks, see as pluses and minus of each approach.
Once we have a more complete listing of these, maybe the decision becomes
more obvious !

Regards
-- Hubertus

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by ebiederm on Tue, 07 Feb 2006 17:20:46 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

>>>I can't think of any real use cases where you would specifically want A)
>>>without B).
>
>> You misrepresent my approach.
> [...]
>
>> Second I am not trying to just implement a form of virtualizing PIDs.
>> Heck I don't intend to virtualize anything. The kernel has already
>> virtualized everything I require. I want to implement multiple
>> instances of the current kernel global namespaces. All I want is
>> to be able to use the same name twice in user space and not have
>> a conflict.

> if you want not virtualize anything, what is this discussion about? :)
> can you provide an URL to your sources? you makes lot's of statements about that
> your network virtualization solution is better/more complete, so I'd like to see
> your solution in whole rather than only words.
> Probably this will help.

Sure.

I think it is more an accident of time, and the fact that I am quite
proud of where I am at. You quite likely have improved things in openvz
since last I looked as well. Currently my code quality is only a proof
of concept but the tree is below.

git://git.kernel.org/pub/scm/linux/kernel/git/ebiederm/linux -2.6-ns.git/

Basically the implementation appears to the user as a separate instance
of the network stack. Since the tree was experimental the path is
absolutely horrible. I am in the process of cleaning and redoing all
of that now in preparation for kernel inclusion.

But I suspect I am in the lead as no one else had noticed the ipv6
reference counting bugs.

>> I disagree with a struct container simply because I do not see what
>> value it happens to bring to the table. I have yet to see a problem
>> that it solves that I have not solved yet.
> again, source would help to understand your solution and problem you solved and
> not solved yet.

Above. But at least with pids it has all been posted on the mailing list.

I think I have solved most of the code structural issues and the big
kernel API issues. A lot of the little things I have not gotten to yet
as I figured it was best approached later.

>> In addition I depart from vserver and other implementations in another
>> regard. It is my impression a lot of their work has been done so
>> those projects are maintainable outside of the kernel, which makes
>> sense as that is where those code bases live. But I don't think that
>> gives the best solution for an in kernel implementation, which is
>> what we are implementing.
> These soltuions are in kernel implementations actually.

Sorry in/out in this context I was referring to the stock linux kernel.
As soon as I had a viable proof of concept I began working to get
my code merged.

Eric

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by serue on Tue, 07 Feb 2006 20:19:08 GMT
View Forum Message <> Reply to Message

Quoting Hubertus Franke (frankeh@watson.ibm.com):
> The vpid approach has the drawbacks of having to identify the conversion
> spots
> of all vpid vs. pid semantics. On the otherhand it does take advantage
> of the fact that no virtualization has to take place until a "container"
> has been migrated, thus rendering most of the vpid<->pid calls to be
> noops.
>
> What I like about the pspace approach is that it explicitely defines in the
> code
> when I am using a different pspace for the lookup. That is kind of hidden in
> the vpid/pid approach.

I agree with this. From a maintenance pov, imagining making a minor change to some pid-related code, if I see something doing effectively "if (pspace1 == pspace2 && pid1==pid2)" that is clear, whereas trying to remember whether I'm supposed to return the pid or vpid can get really confusing. We actually had some errors with that while we were developing the first vpid patchset we posted in december.

I believe that from a vserver point of view, either approach will work. You either create a new pspace and make 'init' pid 1 in that pspace, or, in the openvz approach, you start virtualizing with a hashtable so userspace in the new vserver/container/vz/whateveritscalled sees that init as pid1, while the rest of the system sees it as pid 3270 or something.

Likewise, for checkpoint/restore and migration, either approach works. All we really need is, on restore/migrate, to be able to create processes with their original pids, so we can do that either with real pids in a new container, or virtualized pids faked for process in the same vz.

Are there other uses of pid virtualization which one approach or the other cannot accomodate?

If not, then I for one lean towards the more maintainable code. (which I'm sure we're not agreed on is Eric's, but imvho it is)

-serge

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Hubertus Franke on Tue, 07 Feb 2006 20:46:11 GMT

Serge E. Hallyn wrote:
> Quoting Hubertus Franke (frankeh@watson.ibm.com):
>
>>The vpid approach has the drawbacks of having to identify the conversion
>>spots
>>of all vpid vs. pid semantics. On the otherhand it does take advantage
>>of the fact that no virtualization has to take place until a "container"
>>has been migrated, thus rendering most of the vpid<->pid calls to be
>>noops.
>>
>>What I like about the pspace approach is that it explicitely defines in the
>>code
>>when I am using a different pspace for the lookup. That is kind of hidden in
>>the vpid/pid approach.
>
>
> I agree with this. From a maintenance pov, imagining making a minor
> change to some pid-related code, if I see something doing effectively
> "if (pspace1 == pspace2 && pid1==pid2)" that is clear, whereas trying
> to remember whether I'm supposed to return the pid or vpid can get
> really confusing. We actually had some errors with that while we were
> developing the first vpid patchset we posted in december.
>
> I believe that from a vserver point of view, either approach will work.
> You either create a new pspace and make 'init' pid 1 in that pspace, or,
> in the openvz approach, you start virtualizing with a hashtable so
> userspace in the new vserver/container/vz/whateveritscalled sees that
> init as pid1, while the rest of the system sees it as pid 3270 or
> something.
>
> Likewise, for checkpoint/restore and migration, either approach works.
> All we really need is, on restore/migrate, to be able to create
> processes with their original pids, so we can do that either with real
> pids in a new container, or virtualized pids faked for process in the
> same vz.
>
> Are there other uses of pid virtualization which one approach or the
> other cannot accomodate?

Kirill brought up that VPS can span a cluster..
if so how do you (Kirill) do that? You pre-partition the pids into allocation
ranges for each container?
Eitherway, if this is an important feature, then one needs to look at
how that is achieved in pspace (e.g. mod the pidmap_alloc() function
to take legal ranges into account). Should still be straight forward.

>

> If not, then I for one lean towards the more maintainable code. (which
> I'm sure we're not agreed on is Eric's, but imvho it is)
>
> -serge
>

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by ebiederm on Tue, 07 Feb 2006 22:00:04 GMT
View Forum Message <> Reply to Message

Hubertus Franke <frankeh@watson.ibm.com> writes:

> Kirill brought up that VPS can span a cluster..
> if so how do you (Kirill) do that? You pre-partition the pids into allocation
> ranges for each container?
> Eitherway, if this is an important feature, then one needs to look at
> how that is achieved in pspace (e.g. mod the pidmap_alloc() function
> to take legal ranges into account). Should still be straight forward.

Actually legal ranges already exist in the form of min/max values.
So that is trivial to implement.

Eric

Subject: The issues for agreeing on a virtualization/namespaces implementation.
Posted by ebiederm on Tue, 07 Feb 2006 22:06:51 GMT
View Forum Message <> Reply to Message

I think I can boil the discussion down into some of the fundamental
questions that we are facing.

Currently everyone seems to agree that we need something like
my namespace concept that isolates multiple resources.

We need these for
PIDS
UIDS
SYSVIPC
NETWORK
UTSNAME
FILESYSTEM
etc.

The questions seem to break down into:
1) Where do we put the references to the different namespaces?
- Do we put the references in a struct container that we reference from struct task_struct?
- Do we put the references directly in struct task_struct?

2) What is the syscall interface to create these namespaces?
- Do we add clone flags?
  (Plan 9 style)
- Do we add a syscall (similar to setsid) per namespace?
  (Traditional unix style)?
- Do we in addition add syscalls to manipulate containers generically?

  I don't think having a single system call to create a container and a new
  instance of each namespace is reasonable as that does not give us a
  path into the future when we create yet another namespace.

  If we have one syscall per each namespace why would we need a container
  structure?

3) How do we refer to namespaces and containers when we are not members?
- Do we refer to them indirectly by processes or other objects that
  we can see and are members?
- Do we assign some kind of unique id to the containers?

4) How do we implement each of these namespaces?
  Besides being maintainable are there other constraints?

5) How do we control the resource inside a namespace starting
  from a process that is outside of that namespace?
- The filesystem mount namespace gave an interesting answer.
  So it is quite possible other namespaces will give
  equally interesting and surprising answers.

6) How do we do all of this efficiently without a noticeable impact on
  performance?
- I have already heard concerns that I might be introducing cache
  line bounces and thus increasing tasklist_lock hold time.
  Which on big way systems can be a problem.

7) How do we allow a process inside a container to create containers
  for it's children?
- In general this is trivial but there are a few ugly issues
  here.

I think these are the key questions of the conversation.

Personally so long as we get true namespaces, implemented in a
performant and maintainable way that a process from the inside can't
distinguish from what we have now I have no hard requirements.

Eric

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Hubertus Franke on Tue, 07 Feb 2006 22:19:27 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Hubertus Franke <frankeh@watson.ibm.com> writes:
>
>
>
>>Kirill brought up that VPS can span a cluster..
>>if so how do you (Kirill) do that? You pre-partition the pids into allocation
>>ranges for each container?
>>Eitherway, if this is an important feature, then one needs to look at
>>how that is achieved in pspace (e.g. mod the pidmap_alloc() function
>>to take legal ranges into account). Should still be straight forward.
>
>
> Actually legal ranges already exist in the form of min/max values.
> So that is trivial to implement.
>

Yipp, didn't want to state the obvious, but also give Kirrill a chance
to explain how its done in OpenVZ.

Ultimately, the same "partitioning" that works on vps_info, should work
on pspace.

-- Hubertus

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Sam Vilain on Tue, 07 Feb 2006 22:43:08 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote [note: quoting sections out of order]:
> Sam Vilain <sam@vilain.net> writes:
>>Let's compare approaches of patchsets before the patchsets themselves.

>>It seems to be, should we:
>> A) make a general form of virtualising PIDs, and hope this assists
>>   later virtualisation efforts (Eric's patch)
>>I can't think of any real use cases where you would specifically want A)
>>without B).
> You misrepresent my approach.

ok, after reading more of your post, agreed.

> What user interface to export is a debate worth having.

This is the bit that needs a long period of prototyping and experimental
use IMHO. So in essence, we're agreeing on that point.

> First there is a huge commonality in the code bases between the
> different implementations and I have already gotten preliminary
> acceptance from the vserver developers, that my approach is sane. The
> major difference is what user interface does the kernel export,
> and I posted my user interface.
> Second I am not trying to just implement a form of virtualizing PIDs.
> Heck I don't intend to virtualize anything. The kernel has already
> virtualized everything I require. I want to implement multiple
> instances of the current kernel global namespaces. All I want is
> to be able to use the same name twice in user space and not have
> a conflict.

Right, well, I think our approaches might have more in common than
I previously thought.

Indeed, it seems that at least one of the features of Linux-VServer I am
preparing for consideration for inclusion into Linus' tree are your work
:-).

> Beyond getting multiple instance of all of the kernel namespaces
> (which is the hard requirement for migration) my approach is to
> see what is needed for projects like vserver and vps and see how
> their needs can be met as well.

ok, but the question is - doesn't this just constitute a refactoring
once the stable virtualisation code is in?

I'm just a bit nervous about trying to
refactor-approach-and-concepts-as-we-go.

But look, I'll take a closer look at your patches, and see if I can
merge with you anyhow. Thanks for the git repo!

Sam.

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Sam Vilain on Tue, 07 Feb 2006 22:58:10 GMT
View Forum Message <> Reply to Message

Hubertus Franke wrote:
> The container is just an umbrella object that ties every "virtualized"
> subsystem together.

I like this description; it matches roughly with the concepts as
presented by vserver; there is the process virtualisation (vx_info), and
the network virtualisation (nx_info) of Eric's that has been integrated
to the vserver 2.1.x development branch. However the vx_info has become
the de facto umbrella object space as well. These could almost
certainly be split out without too much pain or incurring major
rethinks.

Sam.

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Hubertus Franke on Tue, 07 Feb 2006 23:18:31 GMT
View Forum Message <> Reply to Message

Sam Vilain wrote:
> Hubertus Franke wrote:
>
>> The container is just an umbrella object that ties every "virtualized"
>> subsystem together.
>
>
> I like this description; it matches roughly with the concepts as
> presented by vserver; there is the process virtualisation (vx_info), and
> the network virtualisation (nx_info) of Eric's that has been integrated
> to the vserver 2.1.x development branch. However the vx_info has become
> the de facto umbrella object space as well. These could almost
> certainly be split out without too much pain or incurring major
> rethinks.
>
> Sam.
>

Agreed.. here are some issued we learned from other projects that had
similar interception points.

Having a central umbrella object (let's stick to the name container)
is useful, but being the only object through which every access has to
pass may have drawbacks..

task->container->pspace->pidmap[offset].page implies potential
cachemisses etc.

If overhead becomes too large, then we can stick (cache) the pointer
additionally in the task struct. But ofcourse that should be carefully
examined on a per subsystem base...

==
Another thing to point out is that container's can have overlaps.

C/R should be a policy thing. So if each "subsystem"

> Quote Eric>>>
> PIDS
> UIDS
> SYSVIPC
> NETWORK
> UTSNAME
> FILESYSTEM

is represented as a NAMESPACE, then one can pick and choose as a
policy how these constitute at a conceptual level as a container.
You want something migratable you better make sure that
container implies unique subsystems.
Maybe you want to nest containers, but only want to create a
separate pidspaces for performance isolation (see planetlab work
with vserver).
So, there are many possibilities, that might make perfect sense
for different desired solutions and it seems with the
clone (CLONE_FLAGS_NSPACE_[PIDS/UIDS/SYS.../FS]) one gets a solution
that is flexible, yet embodies may requirements.....

-- Hubertus

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by Hubertus Franke on Tue, 07 Feb 2006 23:35:52 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> I think I can boil the discussion down into some of the fundamental
> questions that we are facing.
>
Man, bearly can keep up with this email load. Addressed some in
previous thread, but will reiterate under this context.

> Currently everyone seems to agree that we need something like
> my namespace concept that isolates multiple resources.
>
> We need these for
> PIDS
> UIDS
> SYSVIPC
> NETWORK
> UTSNAME
> FILESYSTEM
> etc.
>
> The questions seem to break down into:
> 1) Where do we put the references to the different namespaces?
> - Do we put the references in a struct container that we reference from struct task_struct?
> - Do we put the references directly in struct task_struct?

You "cache" task_struct->container->hotsubsys under task_struct->hotsubsys.
We don't change containers other then at clone time, so no coherency issue here !!!!
Which subsystems pointers to "cache", should be agreed by the experts,
but first approach should always not to cache and go through the container.

>
> 2) What is the syscall interface to create these namespaces?
> - Do we add clone flags?
> (Plan 9 style)

Like that approach .. flexible .. particular when one has well specified namespaces.

> - Do we add a syscall (similar to setsid) per namespace?
> (Traditional unix style)?

Where does that approach end .. what's wrong with doing it at clone() time ?
Mainly the naming issue. Just providing a flag does not give me name.

> - Do we in addition add syscalls to manipulate containers generically?
>
> I don't think having a single system call to create a container and a new
> instance of each namespace is reasonable as that does not give us a
> path into the future when we create yet another namespace.
>
Agreed.
> If we have one syscall per each namespace why would we need a container
> structure?
>
> 3) How do we refer to namespaces and containers when we are not members?
> - Do we refer to them indirectly by processes or other objects that
> we can see and are members?

>   - Do we assign some kind of unique id to the containers?

In containers I simply created an explicite name, which ofcourse colides with the
clone() approach ..
One possibility is to allow associating a name with a namespace.
For instance
int set_namespace_name(long flags, const char *name) /* the once we are using in clone */
{
 if (!flag)
  set name of container associated with current.
 if (flag())
  set the name if only one container is associated with the namespace(s)
  identified .. or some similar rule
}

>
>
> 4) How do we implement each of these namespaces?
>   Besides being maintainable are there other constraints?
>
Good question... at least with PID and FS two are there ..
>
> 5) How do we control the resource inside a namespace starting
>   from a process that is outside of that namespace?
>   - The filesystem mount namespace gave an interesting answer.
>   So it is quite possible other namespaces will give
>   equally interesting and surprising answers.
>
>
> 6) How do we do all of this efficiently without a noticeable impact on
>   performance?
>   - I have already heard concerns that I might be introducing cache
>   line bounces and thus increasing tasklist_lock hold time.
>   Which on big way systems can be a problem.

Possible to split the lock up now.. one for each pidspace ?

>
> 7) How do we allow a process inside a container to create containers
>   for it's children?
>   - In general this is trivial but there are a few ugly issues
>   here.

Speaking of pids only here ...
Does it matter, you just hang all those containers hang of init.
What ever hierarchy they form is external ...

>

> I think these are the key questions of the conversation.
>
>
> Personally so long as we get true namespaces, implemented in a
> performant and maintainable way that a process from the inside can't
> distinguish from what we have now I have no hard requirements.
>
>
> Eric
>

-- Hubertus

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by Alexey Kuznetsov on Wed, 08 Feb 2006 00:43:25 GMT
View Forum Message <> Reply to Message

Hello!

> >2) What is the syscall interface to create these namespaces?
> > - Do we add clone flags?
> > (Plan 9 style)
>
> Like that approach .. flexible .. particular when one has well specified
> namespaces.
>
> > - Do we add a syscall (similar to setsid) per namespace?
> > (Traditional unix style)?
>
> Where does that approach end .. what's wrong with doing it at clone() time ?

That most of those namespaces need a special setup rather than a plain copy?

F.e. what are you going to do with NETWORK namespace? The only valid thing
to do is to prepare a new context and to configure its content (addresses,
routing tables, iptables...) later. So that, in this case it is natural
to inherit the context through clone() and to create new context
with a separate syscall.

Seems, only PID space needs to be setup at clone time. All the rest of
suggested namespaces are more convenient to change with separate syscalls.

I would suggest to combine both approaches. Those namespaces, which can be
naturally copied while clone() (f.e. the best example is already existing
CLONE_NEWNS) deserve a clone() flag. The rest are preserved through clone()
and forked and configured later.

Alexey

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Sam Vilain on Wed, 08 Feb 2006 01:16:01 GMT
View Forum Message <> Reply to Message

Kevin Fox wrote:
>>>The container is just an umbrella object that ties every "virtualized"
>>>subsystem together.
>>I like this description; it matches roughly with the concepts as
>>presented by vserver; there is the process virtualisation (vx_info), and
>>the network virtualisation (nx_info) of Eric's that has been integrated
>>to the vserver 2.1.x development branch. However the vx_info has become
>>the de facto umbrella object space as well. These could almost
>>certainly be split out without too much pain or incurring major
>>rethinks.
> How does all of this tie in with CPU Sets? It seems to me, they have
> something not unlike a container already that supports nesting.

Yes, I saw that. AIUI that's mainly about binding groups of processes
to CPUs, to defeat Amdahl's law when it rears its head. It fits into
the containers model as a hard partitioning feature, but is a lot more
crude than the CPU Token Bucket scheduler in Linux-VServer.

No doubt both CPU allocation strategies will be useful.

Sam.

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Kevin Fox on Wed, 08 Feb 2006 02:08:03 GMT
View Forum Message <> Reply to Message

On Wed, 2006-02-08 at 11:58 +1300, Sam Vilain wrote:
> Hubertus Franke wrote:
> > The container is just an umbrella object that ties every "virtualized"
> > subsystem together.
>
> I like this description; it matches roughly with the concepts as
> presented by vserver; there is the process virtualisation (vx_info), and
> the network virtualisation (nx_info) of Eric's that has been integrated
> to the vserver 2.1.x development branch. However the vx_info has become
> the de facto umbrella object space as well. These could almost
> certainly be split out without too much pain or incurring major
> rethinks.

>
> Sam.

How does all of this tie in with CPU Sets? It seems to me, they have
something not unlike a container already that supports nesting.

Kevin

> -
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by ebiederm on Wed, 08 Feb 2006 02:49:08 GMT
View Forum Message <> Reply to Message

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

> Hello!
>
>> >2) What is the syscall interface to create these namespaces?
>> >   - Do we add clone flags?
>> >   (Plan 9 style)
>>
>> Like that approach .. flexible .. particular when one has well specified
>> namespaces.
>>
>> >   - Do we add a syscall (similar to setsid) per namespace?
>> >   (Traditional unix style)?
>>
>> Where does that approach end .. what's wrong with doing it at clone() time ?
>
> That most of those namespaces need a special setup rather than a plain copy?
>
> F.e. what are you going to do with NETWORK namespace? The only valid thing
> to do is to prepare a new context and to configure its content (addresses,
> routing tables, iptables...) later. So that, in this case it is natural
> to inherit the context through clone() and to create new context
> with a separate syscall.

With a NETWORK namespace what I implemented was that you get a empty
namespace with a loopback interface.

But setting up the namespace from the inside is clearly the sane thing

todo.

> Seems, only PID space needs to be setup at clone time. All the rest of
> suggested namespaces are more convenient to change with separate syscalls.

Actually I think I can setup a PID space in a syscall as well.
It is a little odd that your session, and process group change but since
I was keeping 2 pids on the PID space leader I could easily do it.
The fact that getpid() would start returning 1 might be confusing to a some
processes though so clone seems to be the natural time to do it.

> I would suggest to combine both approaches. Those namespaces, which can be
> naturally copied while clone() (f.e. the best example is already existing
> CLONE_NEWNS) deserve a clone() flag. The rest are preserved through clone()
> and forked and configured later.

Sounds reasonable. We make the decision on a case by case base whatever
make sense for the patch and the namespace.

The only real advantage to clone is you can create a bunch of namespaces
all in one shot. Of course that makes sys_clone a little slower.

Eric

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by serue on Wed, 08 Feb 2006 03:36:33 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> Alexey Kuznetsov <kuznet@ms2.inr.ac.ru> writes:
>
> > Hello!
> >
> >> >2) What is the syscall interface to create these namespaces?
> >> >   - Do we add clone flags?
> >> >   (Plan 9 style)
> >>
> >> Like that approach .. flexible .. particular when one has well specified
> >> namespaces.
> >>
> >> >   - Do we add a syscall (similar to setsid) per namespace?
> >> >   (Traditional unix style)?
> >>
> >> Where does that approach end .. what's wrong with doing it at clone() time ?
> >
> > That most of those namespaces need a special setup rather than a plain copy?

> >
> > F.e. what are you going to do with NETWORK namespace? The only valid thing
> > to do is to prepare a new context and to configure its content (addresses,
> > routing tables, iptables...) later. So that, in this case it is natural
> > to inherit the context through clone() and to create new context
> > with a separate syscall.
>
> With a NETWORK namespace what I implemented was that you get a empty
> namespace with a loopback interface.
>
> But setting up the namespace from the inside is clearly the sane thing
> todo.

What I tried to do in a proof of concept long ago was to have
CLONE_NETNS mean that you get access to all the network devices, but
then you could drop/add them. Conceptually I prefer that to getting an
empty namespace, but I'm not sure whether there's any practical use
where you'd want that...

-serge

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by ebiederm on Wed, 08 Feb 2006 03:52:15 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

>
> What I tried to do in a proof of concept long ago was to have
> CLONE_NETNS mean that you get access to all the network devices, but
> then you could drop/add them. Conceptually I prefer that to getting an
> empty namespace, but I'm not sure whether there's any practical use
> where you'd want that...

My observation was that the network stack does not come out cleanly
as a namespace unless you adopt the rule that a network device
belongs to exactly one network namespace.

With that rule dealing with the network stack is just a matter of making
some currently global variables/data structures per container.

A pain to do the first round but easy to maintain once you are there
and the logic of the code doesn't need to change.

Eric

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Paul Jackson on Wed, 08 Feb 2006 04:21:37 GMT
View Forum Message <> Reply to Message

The driving force for cpusets are NUMA architectures.

Cpusets represent the topologies of NUMA systems, with hierarchies
of cabinets, drawers, boards, packages, cores, hyperthreads, and
with chunks of main memory associated usually with the board, but
sometimes a layer or two up or down.

Since not all cpus have the same access performance (delay and
bandwidth) to all memory chunks (nodes), for optimum performance one
wants to bind tasks, cpus and memory together, so as to run tasks on
sets of cpus and memory that are "near" to each other, and to size the
sets appropriately for the workload presented by the tasks.

Cpusets have no explicit awareness of topology; they just provides a
file system style hierarchy of named, permissioned sets, where each set
has:
  mems - the memory nodes in that set
  cpus - the cpus in that set
  tasks - the tasks running on these cpus and mems

For any cpuset, the 'cpus' and 'mems' are a subset of its parent in the
hierarchy, and the root of the hierarchy (usually mounted at /dev/cpuset)
contains all the online cpus and mems in the system.

--
                I won't rest till it's the best ...
                Programmer, Linux Scalability
                Paul Jackson <pj@sgi.com> 1.925.600.0401

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by Herbert Poetzl on Wed, 08 Feb 2006 04:37:21 GMT
View Forum Message <> Reply to Message

On Tue, Feb 07, 2006 at 08:52:15PM -0700, Eric W. Biederman wrote:
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
> >
> > What I tried to do in a proof of concept long ago was to have
> > CLONE_NETNS mean that you get access to all the network devices, but
> > then you could drop/add them. Conceptually I prefer that to getting an
> > empty namespace, but I'm not sure whether there's any practical use
> > where you'd want that...

>
> My observation was that the network stack does not come out cleanly
> as a namespace unless you adopt the rule that a network device
> belongs to exactly one network namespace.

yep, that's what the first network virtualization for
Linux-VServer aimed at, but found too complicated
the second one uses 'pairs' of communicating devices
to send between guests/host

> With that rule dealing with the network stack is just a matter of
> making some currently global variables/data structures per container.

yep, like the universal loopback and so ...

> A pain to do the first round but easy to maintain once you are there
> and the logic of the code doesn't need to change.

best,
Herbert

> Eric

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by ebiederm on Wed, 08 Feb 2006 04:46:26 GMT
View Forum Message <> Reply to Message

Herbert Poetzl <herbert@13thfloor.at> writes:

> yep, that's what the first network virtualization for
> Linux-VServer aimed at, but found too complicated
> the second one uses 'pairs' of communicating devices
> to send between guests/host

Well you need the pairs of course, to communication between
the two stack ``instances''.

>> With that rule dealing with the network stack is just a matter of
>> making some currently global variables/data structures per container.
>
> yep, like the universal loopback and so ...

:)

Eric

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by Herbert Poetzl on Wed, 08 Feb 2006 04:56:14 GMT
View Forum Message <> Reply to Message

On Tue, Feb 07, 2006 at 03:06:51PM -0700, Eric W. Biederman wrote:
>
> I think I can boil the discussion down into some of the fundamental
> questions that we are facing.
>
> Currently everyone seems to agree that we need something like
> my namespace concept that isolates multiple resources.
>
> We need these for
> PIDS
> UIDS
> SYSVIPC
> NETWORK
> UTSNAME
> FILESYSTEM
> etc.
>
> The questions seem to break down into:
> 1) Where do we put the references to the different namespaces?
> - Do we put the references in a struct container that we
> reference from struct task_struct?

no, just let the tasks be in groups of disjunct spaces
so that they can have shared or private structures for
each of the identified spaces

> - Do we put the references directly in struct task_struct?

yes, IMHO that's the way to do it .. Linux-VServer is
moving in this direction for some time now, but we need
to add a special space for context capabilities and
context flags, basically a context struct, similar to
a namespace ...

> 2) What is the syscall interface to create these namespaces?
> - Do we add clone flags?
> (Plan 9 style)

I'd definitely prefer that, maybe if necessary with a
'new' clone syscall which allows to do a little more
than clone does now, e.g.

 - clone into container/context/guest
 - set space initializations and limits, etc ...

> - Do we add a syscall (similar to setsid) per namespace?
> (Traditional unix style)?

doesn't make sense for the creation, but a syscall
for moving between and management of spaces are very
important ...

> - Do we in addition add syscalls to manipulate containers
> generically?
>
> I don't think having a single system call to create a container
> and a new instance of each namespace is reasonable as that does
> not give us a path into the future when we create yet another
> namespace.
>
> If we have one syscall per each namespace why would we need a
> container structure?

for the beforementioned permissions and flags, but
you can as well see it as separate 'context' space

> 3) How do we refer to namespaces and containers when we are not members?
> - Do we refer to them indirectly by processes or other objects that
> we can see and are members?

the process will be an unique identifier to the
namespace, but it might not be easy to use it, so
IMHO it might at least make sense to ...

> - Do we assign some kind of unique id to the containers?

have an unique identifier for the context space so
that somebody can cherry pick namespaces from there

(in this case, the context space would hold references
to the other namespaces which are the ones used by
new tasks created into a context, basically a template
for them)

> 4) How do we implement each of these namespaces?
> Besides being maintainable are there other constraints?

extensible and with keeping hierarchical structures
in mind .. would be bad if we could not do sub-contexts
without a complete rewrite

> 5) How do we control the resource inside a namespace starting

> from a process that is outside of that namespace?

depends on the resource, but limits have to go to the
context structure, so that they apply for the entire
context space, not just for a task

> - The filesystem mount namespace gave an interesting answer.
> So it is quite possible other namespaces will give
> equally interesting and surprising answers.

> 6) How do we do all of this efficiently without a noticeable impact on
> performance?

> - I have already heard concerns that I might be introducing cache
> line bounces and thus increasing tasklist_lock hold time.
> Which on big way systems can be a problem.

well, we have to be careful with the complexity ...
keep things like refcounting and 'magic' on clone
simple, for the default case ...

> 7) How do we allow a process inside a container to create containers
> for it's children?

> - In general this is trivial but there are a few ugly issues
> here.

a flat implementation will work for a hierarchical
design if certain things are handled properly, just
think resource management for sub-contexts and
changes in the parent's limits ...

> I think these are the key questions of the conversation.
>
>
> Personally so long as we get true namespaces, implemented in a
> performant and maintainable way that a process from the inside can't
> distinguish from what we have now I have no hard requirements.

keep up the good work!

best,
Herbert

> Eric

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by ebiederm on Wed, 08 Feb 2006 05:03:50 GMT
View Forum Message <> Reply to Message

Hubertus Franke <frankeh@watson.ibm.com> writes:
>
> Agreed.. here are some issued we learned from other projects that had
> similar interception points.
>
> Having a central umbrella object (let's stick to the name container)
> is useful, but being the only object through which every access has to
> pass may have drawbacks..
>
> task->container->pspace->pidmap[offset].page implies potential
> cachemisses etc.
>
> If overhead becomes too large, then we can stick (cache) the pointer
> additionally in the task struct. But ofcourse that should be carefully
> examined on a per subsystem base...

Ok. After talking with the vserver guys on IRC. I think grasp the
importance. The key feature is to have a place to put limits and the
like for your entire container. Look at all of the non-signal stuff
in struct signal for an example. The nested namespaces seem to
be just an implementation detail.

For OpenVZ having the other namespaces nested may have some
importance. I haven't gotten their yet.

The task->container->pspace->.... thing feels very awkward to me,
and feels like it increases our chance getting a cache miss.

So I support the concept of a place to put all of the odd little
things like rlimits for containers. But I would like to flatten
it in the task_struct if we can.

Eric

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by ebiederm on Wed, 08 Feb 2006 05:23:15 GMT
View Forum Message <> Reply to Message

Hubertus Franke <frankeh@watson.ibm.com> writes:

> Eric W. Biederman wrote:

>> I think I can boil the discussion down into some of the fundamental
>> questions that we are facing.
>>
> Man, bearly can keep up with this email load. Addressed some in
> previous thread, but will reiterate under this context.

:)

>> Currently everyone seems to agree that we need something like
>> my namespace concept that isolates multiple resources.
>> We need these for PIDS
>> UIDS
>> SYSVIPC
>> NETWORK
>> UTSNAME
>> FILESYSTEM
>> etc.
>> The questions seem to break down into:
>> 1) Where do we put the references to the different namespaces?
>> - Do we put the references in a struct container that we reference from struct
> task_struct?
>> - Do we put the references directly in struct task_struct?
>
> You "cache" task_struct->container->hotsubsys under task_struct->hotsubsys.
> We don't change containers other then at clone time, so no coherency issue here
> !!!!
> Which subsystems pointers to "cache", should be agreed by the experts,
> but first approach should always not to cache and go through the container.
>
>> 2) What is the syscall interface to create these namespaces?
>> - Do we add clone flags? (Plan 9 style)
>
> Like that approach .. flexible .. particular when one has well specified
> namespaces.
>
>> - Do we add a syscall (similar to setsid) per namespace?
>> (Traditional unix style)?
>
> Where does that approach end .. what's wrong with doing it at clone() time ?
> Mainly the naming issue. Just providing a flag does not give me name.

It really is a fairly even toss up. The usual argument for doing it
this way is that you will get a endless stream of arguments added to
fork+exec other wise. Look of posix_spawn or the windows version if
you want an example. Bits to clone are skirting the edge of a slippery
slope.

>> 3) How do we refer to namespaces and containers when we are not members?

>>   - Do we refer to them indirectly by processes or other objects that
>>   we can see and are members?
>>   - Do we assign some kind of unique id to the containers?
>
> In containers I simply created an explicite name, which ofcourse colides with
> the
> clone() approach ..
> One possibility is to allow associating a name with a namespace.
> For instance
> int set_namespace_name(long flags, const char *name) /* the once we are using
> in clone */
> {
> if (!flag)
>   set name of container associated with current.
> if (flag())
>   set the name if only one container is associated with the
> namespace(s)
>   identified .. or some similar rule
> }
>

What I have done which seems easier than creating new names is to refer
to the process which has the namespace I want to manipulate.

>> 6) How do we do all of this efficiently without a noticeable impact on
>>   performance?
>>   - I have already heard concerns that I might be introducing cache
>>   line bounces and thus increasing tasklist_lock hold time.
>>   Which on big way systems can be a problem.
>
> Possible to split the lock up now.. one for each pidspace ?

At the moment it is worth thinking about. If the problem isn't
so bad that people aren't actively working on it we don't have to
solve the problem for a little while, just be aware of it.

>> 7) How do we allow a process inside a container to create containers
>>   for it's children?
>>   - In general this is trivial but there are a few ugly issues
>>   here.
>
> Speaking of pids only here ...
> Does it matter, you just hang all those containers hang of init.
> What ever hierarchy they form is external ...

In general it is simple. For resource accounting, and for naming so
you can migrate a container with a nested container it is a question
you need to be slightly careful with.

Eric

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Hubertus Franke on Wed, 08 Feb 2006 14:13:25 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Hubertus Franke <frankeh@watson.ibm.com> writes:
>
>>Agreed.. here are some issued we learned from other projects that had
>>similar interception points.
>>
>>Having a central umbrella object (let's stick to the name container)
>>is useful, but being the only object through which every access has to
>>pass may have drawbacks..
>>
>>task->container->pspace->pidmap[offset].page implies potential
>>cachemisses etc.
>>
>>If overhead becomes too large, then we can stick (cache) the pointer
>>additionally in the task struct. But ofcourse that should be carefully
>>examined on a per subsystem base...
>
>
> Ok. After talking with the vserver guys on IRC. I think grasp the
> importance. The key feature is to have a place to put limits and the
> like for your entire container. Look at all of the non-signal stuff
> in struct signal for an example. The nested namespaces seem to
> be just an implementation detail.
>
> For OpenVZ having the other namespaces nested may have some
> importance. I haven't gotten their yet.
>
> The task->container->pspace->.... thing feels very awkward to me,
> and feels like it increases our chance getting a cache miss.
>
> So I support the concept of a place to put all of the odd little
> things like rlimits for containers. But I would like to flatten
> it in the task_struct if we can.
>

My point was to mainly identify the performance culprits and provide
an direct access to those "namespaces" for performance reasons.
So we all agreed on that we need to do that..

Question now (see other's note as well), should we provide

a pointer to each and every namespace in struct task.
Seem rather wasteful to me as certain path/namespaces are not
exercise heavily.

Having one object "struct container" that still embodies all
namespace still seems a reasonable idea.
Otherwise identifying the respective namespace of subsystems will
have to go through container->init->subsys_namespace or similar.
Not necessarily bad either..

-- Hubertus

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by serue on Wed, 08 Feb 2006 14:38:13 GMT
View Forum Message <> Reply to Message

Quoting Herbert Poetzl (herbert@13thfloor.at):
> > 3) How do we refer to namespaces and containers when we are not members?
> > - Do we refer to them indirectly by processes or other objects that
> > we can see and are members?
>
> the process will be an unique identifier to the
> namespace, but it might not be easy to use it, so
> IMHO it might at least make sense to ...

Especially from userspace. If I want to start a checkpoint on a
container, but I have to use the process to identify the
container/namespace, well I can't uniquely specify the process by pid
anymore...

-serge

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by Hubertus Franke on Wed, 08 Feb 2006 14:40:17 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Hubertus Franke <frankeh@watson.ibm.com> writes:
>
>
>>Eric W. Biederman wrote:
>>

>>>2) What is the syscall interface to create these namespaces?
>>> - Do we add clone flags? (Plan 9 style)
>>
>>Like that approach .. flexible .. particular when one has well specified
>>namespaces.
>>
>>
>>> - Do we add a syscall (similar to setsid) per namespace?
>>> (Traditional unix style)?
>>
>>Where does that approach end .. what's wrong with doing it at clone() time ?
>>Mainly the naming issue. Just providing a flag does not give me name.
>
>
> It really is a fairly even toss up. The usual argument for doing it
> this way is that you will get a endless stream of arguments added to
> fork+exec other wise. Look of posix_spawn or the windows version if
> you want an example. Bits to clone are skirting the edge of a slippery
> slope.
>

So it seems the clone(flags) is a reasonable approach to create new
namespaces. Question is what is the initial state of each namespace?
In pidspace we know we should be creating an empty pidmap !
In network, someone suggested creating a loopback device
In uts, create "localhost"
Are there examples where we rather inherit ? Filesystem ?
Can we iterate the assumption for each subsystem what people thing is right?

IMHO, there is only a need to refer to a namespace from the global context.
Since one will be moving into a new container, but getting out of one
could be prohibitive (e.g. after migration)
It does not make sense therefore to know the name of a namespace in
a different container.

The example you used below by using the pid comes natural, because
that already limits visibility.

I am still struggling with why we need new sys_calls.
sys_calls already exist for changing certain system parameters (e.g. utsname)
so to me it boils down to identifying a proper initial state when the
namespace is created.

>
>>>3) How do we refer to namespaces and containers when we are not members?
>>> - Do we refer to them indirectly by processes or other objects that
>>> we can see and are members?
>>> - Do we assign some kind of unique id to the containers?

>>
>>In containers I simply created an explicite name, which ofcourse colides with
>>the
>>clone() approach ..
>>One possibility is to allow associating a name with a namespace.
>>For instance
>>int set_namespace_name(long flags, const char *name) /* the once we are using
>>in clone */
>>{
>> if (!flag)
>> set name of container associated with current.
>> if (flag())
>> set the name if only one container is associated with the
>>namespace(s)
>> identified .. or some similar rule
>>}
>>
>
>
> What I have done which seems easier than creating new names is to refer
> to the process which has the namespace I want to manipulate.

Is then the idea to only allow the container->init to manipulate
or is there need to allow other priviliged processes to perform namespace
manipulation?
Also after thinking about it.. why is there a need to have an external name
for a namespace ?

>
>
>>>6) How do we do all of this efficiently without a noticeable impact on
>>>   performance?
>>>   - I have already heard concerns that I might be introducing cache
>>>   line bounces and thus increasing tasklist_lock hold time.
>>>   Which on big way systems can be a problem.
>>
>>Possible to split the lock up now.. one for each pidspace ?
>
>
> At the moment it is worth thinking about. If the problem isn't
> so bad that people aren't actively working on it we don't have to
> solve the problem for a little while, just be aware of it.
>

Agree, just need to be sure we can split it up. But you already keep
a task list per pid-namespace, so there should be no problem IMHO.
If so let's do it now and take it of the table it its as simple as

task_list_lock ::= pspace->task_list_lock

>
>>>7) How do we allow a process inside a container to create containers
>>> for it's children?
>>> - In general this is trivial but there are a few ugly issues
>>>   here.
>>
>>Speaking of pids only here ...
>>Does it matter, you just hang all those containers hang of init.
>>What ever hierarchy they form is external ...
>
>
> In general it is simple. For resource accounting, and for naming so
> you can migrate a container with a nested container it is a question
> you need to be slightly careful with.

Absolutely, that's why it is useful to have an "external" idea of how
containers are constructed of basic namespaces==subsystems.
The it "simply" becomes a policy. E.g. one can not migrate a container
that has shared subsystems.
Resource accounting I agree, that might required active aggregation
at request time.

-- Hubertus

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by Hubertus Franke on Wed, 08 Feb 2006 14:51:03 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Herbert Poetzl (herbert@13thfloor.at):
>
>>>3) How do we refer to namespaces and containers when we are not members?
>>> - Do we refer to them indirectly by processes or other objects that
>>>   we can see and are members?
>>
>>the process will be an unique identifier to the
>>namespace, but it might not be easy to use it, so
>>IMHO it might at least make sense to ...
>
>
> Especially from userspace. If I want to start a checkpoint on a
> container, but I have to use the process to identify the
> container/namespace, well I can't uniquely specify the process by pid
> anymore...

>
> -serge
>

Well we first can agree that througout all processes/tasks of a container
the namespaces used are the same.
Hence looking at the init_task of each container is sufficient.

Restricting visibility to the default container makes sense to me,
because one is not supposed to be able to look into another container.

However, in the global context we do have pid that we can use.

-- Hubertus

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by serue on Wed, 08 Feb 2006 15:17:26 GMT
View Forum Message <> Reply to Message

Quoting Hubertus Franke (frankeh@watson.ibm.com):
> Eric W. Biederman wrote:
> So it seems the clone(flags) is a reasonable approach to create new
> namespaces. Question is what is the initial state of each namespace?
> In pidspace we know we should be creating an empty pidmap !
> In network, someone suggested creating a loopback device
> In uts, create "localhost"
> Are there examples where we rather inherit ? Filesystem ?

Of course filesystem is already implemented, and does inheret a full
copy.

-serge

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by dev on Wed, 08 Feb 2006 15:34:41 GMT
View Forum Message <> Reply to Message

>>Eric W. Biederman wrote:
>>So it seems the clone(flags) is a reasonable approach to create new
>>namespaces. Question is what is the initial state of each namespace?
>>In pidspace we know we should be creating an empty pidmap !
>>In network, someone suggested creating a loopback device
>>In uts, create "localhost"

>>Are there examples where we rather inherit ? Filesystem ?
> Of course filesystem is already implemented, and does inheret a full
> copy.

why do we want to use clone()? Just because of its name and flags?
I think it is really strange to fork() to create network context. What
has process creation has to do with it?

After all these clone()'s are called, some management actions from host
system are still required, to add these IPs/routings/etc.
So? Why mess it up? Why not create a separate clean interface for
container management?

Kirill

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by dev on Wed, 08 Feb 2006 15:35:10 GMT
View Forum Message <> Reply to Message

> The pid-namespace (pspace) provides an approach of fully separate
> the allocation and maintenance of the pids and treating the <pspace,pid>
> tuple as an entity to uniquely identify a task and vice versa.
> As a result the logic of lookup can be pushed down the find_task_by_pid()
> lookup. There are specific cases where the init_task of a container or
> pspace needs to be checked to ensure that signals/waits and alike are
> properly
> handled across pspace boundaries. I think this is an intuitive and clean
> way.
> It also completely avoids the problem of having to think about all the
> locations
> at the user/kernel boundary where a vpid/pid conversion needs to take
> place.
> It also avoids the problems that logically vpids and pids are different
> types and
> therefore it would have been good to have type checking automatically
> identify
> problem spots.
> On the negative side, it does require to maintain a pidmap per pidspace.
Additional negative sides:
- full isolation can be inconvinient from containers management point of
view. You will need to introduce new modified tools such as top/ps/kill
and many many others. You won't be able to strace/gdb processes from the
host also.
- overhead when virtualization is off, result is not the same.
- additional args everywhere (stack usage, etc.)

> The vpid approach has the drawbacks of having to identify the conversion

> spots
> of all vpid vs. pid semantics. On the otherhand it does take advantage
> of the fact that no virtualization has to take place until a "container"
> has been migrated, thus rendering most of the vpid<->pid calls to be
> noops.
It has some other additional advantages:
- flexible: you can select full isolation or weak is required. I really
believe this is very important.

> The container is just an umbrella object that ties every "virtualized"
> subsystem
> together.
Yep. And containers were what I wanted to start with actually. Not VPIDs.

Kirill

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by dev on Wed, 08 Feb 2006 15:42:05 GMT
View Forum Message <> Reply to Message

> My point was to mainly identify the performance culprits and provide
> an direct access to those "namespaces" for performance reasons.
> So we all agreed on that we need to do that..
After having looked at Eric's patch, I can tell that he does all these
dereferences in quite the same amount.

For example, lot's of skb->sk->host->...
while in OpenVZ it would be econtainer()->... which is essentially
current->container->...

So until someone did measurements it looks doubtfull that one solution
is better than the another.

> Question now (see other's note as well), should we provide
> a pointer to each and every namespace in struct task.
> Seem rather wasteful to me as certain path/namespaces are not
> exercise heavily.

> Having one object "struct container" that still embodies all
> namespace still seems a reasonable idea.
> Otherwise identifying the respective namespace of subsystems will
> have to go through container->init->subsys_namespace or similar.
> Not necessarily bad either..

why not simply container->subsys_namespace?

Kirill

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by Hubertus Franke on Wed, 08 Feb 2006 15:57:22 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
>>> Eric W. Biederman wrote:
>>> So it seems the clone(flags) is a reasonable approach to create new
>>> namespaces. Question is what is the initial state of each namespace?
>>> In pidspace we know we should be creating an empty pidmap !
>>> In network, someone suggested creating a loopback device
>>> In uts, create "localhost"
>>> Are there examples where we rather inherit ? Filesystem ?
>>
>> Of course filesystem is already implemented, and does inheret a full
>> copy.
>
>
> why do we want to use clone()? Just because of its name and flags?
> I think it is really strange to fork() to create network context. What
> has process creation has to do with it?
>
> After all these clone()'s are called, some management actions from host
> system are still required, to add these IPs/routings/etc.
> So? Why mess it up? Why not create a separate clean interface for
> container management?
>
> Kirill
>

We need a "init" per container, which represents the context of the
system represented by the container.
If that is the case, then why not create the container such that
we specify what namespaces need to be new for a container at
the container creation time and initialize them to a well understood
state that makes sense (e.g. copy namespace (FS, uts) , new fresh state (pid)).

Then use the standard syscall to modify state (now "virtualized" through
the task->xxx_namespace access).

Do you see a need to change the namespace of a container after it
has been created. I am not referring to the state of the namespace
but truely moving to a completely different namespace after the
container has been created.

Obviously you seem to have some other usage in mind, beyond what my
limited vision can see. Can you share some of those examples, because
that would help this discussion along ...

Thanks a 10^6.

-- Hubertus

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by ebiederm on Wed, 08 Feb 2006 16:39:51 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

>> My point was to mainly identify the performance culprits and provide
>> an direct access to those "namespaces" for performance reasons.
>> So we all agreed on that we need to do that..
> After having looked at Eric's patch, I can tell that he does all these
> dereferences in quite the same amount.
>
> For example, lot's of skb->sk->host->...
> while in OpenVZ it would be econtainer()->... which is essentially
> current->container->...

Except at that point in time I cannot use current, because it
does not have necessarily have the appropriate context. So doubt
you could correctly use econtainer there.

> So until someone did measurements it looks doubtfull that one solution is better
> than the another.

I agree, exactly where we look is a minor matter, unless we can find
an instance where there are good reasons for preferring one representation
over another. Performance currently does not look a sufficient reason.

My basis for preferring a flat layout is essentially because
that is how other similar interfaces are currently implemented
in the kernel.

In linux we have a plan9 inspired system call called clone.
One of the ideas with clone is that when you create a new
task you either share or you don't share resources. Namespaces
are one of those resources.

Since we are dealing with concepts that appear to fit the
existing model beautifully my feeling is to use the existing
model of how things are currently implemented in the kernel.

The only other reason I can think of is namespace implementation

independence. If we have an additional structure that we collect
up the pointers it feels like it causes unnecessary tying.

The best justification I can think of for putting it all in
one structure seems like the current->container current->econtainer
thing that comes out of OpenVZ. Currently I have followed the
threads enough to know it exists but I yet understand it.

If there is a good reason for the container/econtainer thing I can
certainly seem some benefit.

Eric

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by ebiederm on Wed, 08 Feb 2006 16:48:14 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

>>>Eric W. Biederman wrote:
>>>So it seems the clone(flags) is a reasonable approach to create new
>>>namespaces. Question is what is the initial state of each namespace?
>>>In pidspace we know we should be creating an empty pidmap !
>>>In network, someone suggested creating a loopback device
>>>In uts, create "localhost"
>>>Are there examples where we rather inherit ? Filesystem ?
>> Of course filesystem is already implemented, and does inheret a full
>> copy.
>
> why do we want to use clone()? Just because of its name and flags?
> I think it is really strange to fork() to create network context. What has
> process creation has to do with it?

Agreed. Although clones brother unshare takes process creation out of the
picture, but otherwise preserves the same interface.

> After all these clone()'s are called, some management actions from host system
> are still required, to add these IPs/routings/etc.
> So? Why mess it up? Why not create a separate clean interface for container
> management?

If we need additional arguments besides create the thing. We have a clear
argument that clone is completely the wrong interface.

However. So far I have not seen an instance where using the existing
standard configuration mechanisms from inside the namespace is not the
proper way to set things up. The only thing I know that needs to happen from
outside is to pass the container a network interface. And if it is a physical
interface that is all that must happen.

Eric

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by ebiederm on Wed, 08 Feb 2006 17:16:45 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

> Additional negative sides:
> - full isolation can be inconvinient from containers management point of
> view. You will need to introduce new modified tools such as top/ps/kill and many
> many others. You won't be able to strace/gdb processes from the host also.

Ok. I have to pick a nit here. Needing all new tools top/ps/kill was an artifact
of your implementation. Mine does not suffer from it.

> - overhead when virtualization is off, result is not the same.
> - additional args everywhere (stack usage, etc.)

Agreed. When using a PID namespace the code always behaves the same.
Which is with more arguments than the code used to have.
However the code always behaving the same is a tremendous advantage
for maintainability.

>> The vpid approach has the drawbacks of having to identify the conversion spots
>> of all vpid vs. pid semantics. On the otherhand it does take advantage
>> of the fact that no virtualization has to take place until a "container"
>> has been migrated, thus rendering most of the vpid<->pid calls to be
>> noops.
> It has some other additional advantages:
> - flexible: you can select full isolation or weak is required. I really believe
> this is very important.

I am willing to be convinced but so far there seem to be other solutions
like running gdb_stubs inside your container. To address most of the issues.

In a fairly real sense mucking with a container from the outside appears
to be a sysadmin layering violation.

>> The container is just an umbrella object that ties every "virtualized"
>> subsystem

>> together.
> Yep. And containers were what I wanted to start with actually. Not VPIDs.

The historical linux approach is to build things out of tasks sharing
resources that give the impression of containers not out containers
and their children. I am still trying to understand why that model
does not work in this instance.

Eric

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by ebiederm on Wed, 08 Feb 2006 17:46:14 GMT
View Forum Message <> Reply to Message

Hubertus Franke <frankeh@watson.ibm.com> writes:

>
> So it seems the clone(flags) is a reasonable approach to create new
> namespaces. Question is what is the initial state of each namespace?
> In pidspace we know we should be creating an empty pidmap !
> In network, someone suggested creating a loopback device
> In uts, create "localhost"
"localhost" is wrong. Either copy or set it to the value from
system initialization time.

> Are there examples where we rather inherit ? Filesystem ?
> Can we iterate the assumption for each subsystem what people thing is right?

I think this needs to happen on a pure subsystem basis as we merge the
patches.

> IMHO, there is only a need to refer to a namespace from the global context.
> Since one will be moving into a new container, but getting out of one
> could be prohibitive (e.g. after migration)
> It does not make sense therefore to know the name of a namespace in
> a different container.
>
> The example you used below by using the pid comes natural, because
> that already limits visibility.
>
> I am still struggling with why we need new sys_calls.
> sys_calls already exist for changing certain system parameters (e.g. utsname)
> so to me it boils down to identifying a proper initial state when the
> namespace is created.

Agreed. But we can't always count on everything having a useful state

that we can modify from the inside. So it is important to leave the
option open at least for those case.

And again there is always the idea that by adding flags we will
transform fork or fork+exec into a spaghetti system call. I think
that is a reasonable concern.

Also there is always the danger that we run out of clone flags.

>> What I have done which seems easier than creating new names is to refer
>> to the process which has the namespace I want to manipulate.
>
> Is then the idea to only allow the container->init to manipulate
> or is there need to allow other priviliged processes to perform namespace
> manipulation?
> Also after thinking about it.. why is there a need to have an external name
> for a namespace ?

Largely it connects to the super chroot usage where you have one
sysadmin he has multiple daemons each running in their own environment
for isolation purposes. Nothing is installed in the chroot so an
attacker that gets in cannot do anything.

>>>>6) How do we do all of this efficiently without a noticeable impact on
>>>>   performance?
>>>>   - I have already heard concerns that I might be introducing cache
>>>>   line bounces and thus increasing tasklist_lock hold time.
>>>>   Which on big way systems can be a problem.
>>>
>>>Possible to split the lock up now.. one for each pidspace ?
>> At the moment it is worth thinking about. If the problem isn't
>> so bad that people aren't actively working on it we don't have to
>> solve the problem for a little while, just be aware of it.
>>
>
> Agree, just need to be sure we can split it up. But you already keep
> a task list per pid-namespace, so there should be no problem IMHO.
> If so let's do it now and take it of the table it its as simple as
>
> task_list_lock ::= pspace->task_list_lock

Actually I don't although that could be trivial. But it is the
wrong split. The problem is that it is a lock with global effect.

Eric

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by serue on Wed, 08 Feb 2006 18:03:09 GMT
View Forum Message <> Reply to Message

Quoting Hubertus Franke (frankeh@watson.ibm.com):
> IMHO, there is only a need to refer to a namespace from the global context.
> Since one will be moving into a new container, but getting out of one
> could be prohibitive (e.g. after migration)
> It does not make sense therefore to know the name of a namespace in
> a different container.

Not sure I agree. What if we are using a private namespace for a vserver, and then we want to create a private namespace in there for a mobile application. Since we're talking about nested namespaces, this should be possible.

Now I believe Eric's code so far would make it so that you can only refer to a namespace from it's *creating* context. Still restrictive, but seems acceptable.

(right?)

-serge

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by Hubertus Franke on Wed, 08 Feb 2006 18:31:08 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Hubertus Franke (frankeh@watson.ibm.com):
>
>>IMHO, there is only a need to refer to a namespace from the global context.
>>Since one will be moving into a new container, but getting out of one
>>could be prohibitive (e.g. after migration)
>>It does not make sense therefore to know the name of a namespace in
>>a different container.
>
>
> Not sure I agree. What if we are using a private namespace for a
> vserver, and then we want to create a private namespace in there for a
> mobile application. Since we're talking about nested namespaces, this
> should be possible.
>
> Now I believe Eric's code so far would make it so that you can only
> refer to a namespace from it's *creating* context. Still restrictive,

> but seems acceptable.
>

That's what I meant .. as usually used the wrong word..
s/global context/spawing context/g .. because that's the only
place where you have a pid to refer to the newly created container !

> (right?)
>

Yes, seriii .. ahmm serue
> -serge
>

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by Herbert Poetzl on Wed, 08 Feb 2006 19:02:30 GMT
View Forum Message <> Reply to Message

On Wed, Feb 08, 2006 at 10:57:24AM -0500, Hubertus Franke wrote:
>Kirill Korotaev wrote:
>>>>Eric W. Biederman wrote:
>>>>So it seems the clone(flags) is a reasonable approach to create new
>>>>namespaces. Question is what is the initial state of each namespace?
>>>>In pidspace we know we should be creating an empty pidmap !
>>>>In network, someone suggested creating a loopback device
>>>>In uts, create "localhost"
>>>>Are there examples where we rather inherit ? Filesystem ?
>>>
>>>Of course filesystem is already implemented, and does inheret a full
>>>copy.

I try to comment on both mails here because I thing that
clone() is basically a good interface, but will require
some redesign and/or extension ...

>> why do we want to use clone()?

because it is a natural and existing interface for this purpose
at least in Linux-VServer it would work (to some extend). why?

because we aleady use tools like chbind and chcontext which
do similar things as chroot, and chroot could, in theory, use
clone() and rbind to do it's job ...

>> Just because of its name and flags?

extending the flags seems natural to me, but the problem might
actually be that there are not enough of them left

>> I think it is really strange to fork() to create network context.

if you look at it as network namespace and sharing existing
spaces and/or creating new ones, then clone() and unshare()
make pretty much sense there ...

>> What has process creation has to do with it?

it is a natural interface where you can decide whether to
share a space or acquire a new one ... IMHO it would make
sense to get trivial userspace tools to create those
new spaces in one go, so that the user can use those
'building blocks' to create new spaces whenever she needs

>> After all these clone()'s are called, some management actions
>> from host system are still required, to add these IPs/routings/etc.

not necessarily, for example Linux-VServer uses some kind
of 'priviledged' mode, in which the initial guest process
can modify various things (like in this case the networking)
and setup whatever is required, then, shortly after, giving
up those priviledges ...

>> So? Why mess it up?
>> Why not create a separate clean interface for container management?

I'm not against a clean interface at all, but how would
such a 'clean' interface look like?

- a complicated sysfs interface where you write strange
  values into even stranger places?
- 40 different syscalls to do stuff like adding or removing
  various parts from the spaces?
- a new ioctl for processes?

>> Kirill
>
> We need a "init" per container, which represents the context of the
> system represented by the container.

that's only partially true, for example Linux-VServer also
allows for light-weight guests/containers which do not
have a separate init process, just a 'fake' one, so we can
save the resources consumed by a separate init process ...

it turns out that this works perfectly fine, even without
that fake init, if you teach a few tools like pstree that
they should not blindly assume that there is a pid=1 :)

> If that is the case, then why not create the container such that
> we specify what namespaces need to be new for a container at the
> container creation time and initialize them to a well understood
> state that makes sense (e.g. copy namespace (FS, uts) , new fresh
> state (pid)).

agreed, but now comes the interesting part, how does such
a well understood state look like for all contexts?

obviously the name space makes a complete copy, leading to
various issues when you 'try' to get rid of the 'copied'
data, like host filesystem mount points and such ...

removing the mounts 'above' a certain chroot() path might
seem like a good solution here, but actually it will cause
issues when you want to maintain/access a guest/container
from the host/parent

leaving all mounts in place will require to either inherit
changes from the host/parent to all guests/containers, just
to avoid having e.g. /mnt/cdrom mounted in process A, which
does not even see it (as it's accessible space starts
somewhere else) and therefore being unable to eject the
thing, although it is obviously unused

> Then use the standard syscall to modify state (now "virtualized"
> through the task->xxx_namespace access).

works as long as you have a handle for that, and actually
you do not need one init per guest/container, you need
one 'uniquely identified' task 'outside' the container

which in the typical case already makes two of them, the
'handle' task outside and the 'init' task inside ...

> Do you see a need to change the namespace of a container after it
> has been created. I am not referring to the state of the namespace
> but truely moving to a completely different namespace after the
> container has been created.

container itself, probably not, task, definitely yes ...
i.e. you definitely want to move between all those spaces
given that you are sufficiently priviledged, which is a
completely different can of worms ...

> Obviously you seem to have some other usage in mind, beyond what my
> limited vision can see. Can you share some of those examples, because
> that would help this discussion along ...

I guess one case where the separate container setup is
desired is when you want to keep a container alive even
after the processes have seized to exist. for example to
visit a 'stopped' guest/context just to do some emergency
operations or install some files/packages/etc

IMHO this can easily be solved by keeping the 'handle'
process (or whatever handle may be used for the complete
context consisting of all spaces) around, even if no
process is using those spaces ...

the context as 'collection' of certain namespaces is
definitely something which will be required to allow to
'move' into a specific container from the host/parent
side, as the parent process obviously does not hold that
information.

best,
Herbert

> Thanks a 10^6.
>
> -- Hubertus
>
>
>

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by Dave Hansen on Wed, 08 Feb 2006 20:21:39 GMT
View Forum Message <> Reply to Message

On Wed, 2006-02-08 at 12:03 -0600, Serge E. Hallyn wrote:
> Now I believe Eric's code so far would make it so that you can only
> refer to a namespace from it's *creating* context. Still restrictive,
> but seems acceptable.

The same goes for filesystem namespaces. You can't see into random
namespaces, just the ones underneath your own. Sounds really reasonable
to me.

-- Dave

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Dave Hansen on Wed, 08 Feb 2006 20:43:55 GMT
View Forum Message <> Reply to Message

On Wed, 2006-02-08 at 18:36 +0300, Kirill Korotaev wrote:
> - full isolation can be inconvinient from containers management point of
> view. You will need to introduce new modified tools such as top/ps/kill
> and many many others. You won't be able to strace/gdb processes from the
> host also.

I'd like to put a theory out there: the more isolation we perform, the easier checkpointing and migration become to guarantee.

Agree? Disagree?

But, full isolation is hard to code. The right approach is very likely somewhere in the middle where we require some things to happen underneath us. For instance, requiring that the filesystem be made consistent if a container is moved across systems.

-- Dave

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by ebiederm on Wed, 08 Feb 2006 21:04:14 GMT
View Forum Message <> Reply to Message

Dave Hansen <haveblue@us.ibm.com> writes:

> On Wed, 2006-02-08 at 18:36 +0300, Kirill Korotaev wrote:
>> - full isolation can be inconvinient from containers management point of
>> view. You will need to introduce new modified tools such as top/ps/kill
>> and many many others. You won't be able to strace/gdb processes from the
>> host also.
>
> I'd like to put a theory out there: the more isolation we perform, the
> easier checkpointing and migration become to guarantee.
>
> Agree? Disagree?

Agree. But that does not address the reasons OpenVZ and Vserver exist.

> But, full isolation is hard to code.

Disagree. If you limit your self to just changing the code that translates from names to objects it is a very narrow slice of code, and there are very few surprises. There is a lot of grunt work involved but it easy to tell if you got everything and did it correctly.

Other approaches are more adhoc, take short cuts, and seem prone to missing
the corner cases.

> The right approach is very likely
> somewhere in the middle where we require some things to happen
> underneath us. For instance, requiring that the filesystem be made
> consistent if a container is moved across systems.

Possibly. That is very out from where we are at the moment.
Let's get the isolation and see where we are at.

Eric

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by serue on Wed, 08 Feb 2006 21:22:12 GMT
View Forum Message <> Reply to Message

Quoting Dave Hansen (haveblue@us.ibm.com):
> On Wed, 2006-02-08 at 12:03 -0600, Serge E. Hallyn wrote:
> > Now I believe Eric's code so far would make it so that you can only
> > refer to a namespace from it's *creating* context. Still restrictive,
> > but seems acceptable.
>
> The same goes for filesystem namespaces. You can't see into random
> namespaces, just the ones underneath your own. Sounds really reasonable
> to me.

Hmmm? I suspect I'm misreading what you're saying, but to be clear:

Let's say I start a screen session. In one of those shells, I clone,
specify CLONE_NEWNS, and exec a shell. now i do a bunch of mounting.
Other shells in the screen session won't see the results of those
mounts, and if i ctrl-d, the shell which started the screen session
can't either. Each of these is in the "parent filesystem namespace".

OTOH, shared subtrees specified in the parent shell could make it such
that the parent ns, but not others, see the results. Is that what
you're referring to?

thanks,
-serge

Subject: Re: The issues for agreeing on a virtualization/namespaces

implementation.
Posted by ebiederm on Wed, 08 Feb 2006 22:28:12 GMT
View Forum Message <> Reply to Message

Hubertus Franke <frankeh@watson.ibm.com> writes:

> Eric W. Biederman wrote:
>> Hubertus Franke <frankeh@watson.ibm.com> writes:
>>
>>>Eric W. Biederman wrote:
>>>
>>
>>>>3) How do we refer to namespaces and containers when we are not members?
>>>> - Do we refer to them indirectly by processes or other objects that
>>>>   we can see and are members?
>>>> - Do we assign some kind of unique id to the containers?
>>>
>>>
>> What I have done which seems easier than creating new names is to refer
>> to the process which has the namespace I want to manipulate.
>
> Is then the idea to only allow the container->init to manipulate
> or is there need to allow other priviliged processes to perform namespace
> manipulation?
> Also after thinking about it.. why is there a need to have an external name
> for a namespace ?

There are several cases.

Passing network devices to a childs namespace, as usually
the loopback interface is not enough.

Monitoring the namespace from outside, so among other things
you aren't required to checkpoint and migrate your monitoring
daemon.

There are several other control and monitoring operations
that I am not quite as familiar. One of them is the
vserver idea of entering a guest.

To expand on things a little bit. If we have interfaces
that take strings we can refer to an arbitrary child process
as pid/pid/pid/.... So we should not be limited to what
is at the init of the container. If that proves desirable.

Permissions checks for most of these operations require some
serious thinking before they are merged.

Eric

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by ebiederm on Thu, 09 Feb 2006 00:24:16 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@openvz.org> writes:

> Hello,
>
> I tried to take into account all the comments from you guys (thanks a lot for
> them!) and prepared a new version of virtualization patches. I will send only 4
> patches today, just not to overflow everyone and keep it clear/tidy/possible to
> review.
>
> This patch introduces some abstract container kernel structure and a number of
> operations on it.
>
> The important properties of the proposed container implementation:
> - each container has unique ID in the system
> - each process in the kernel can belong to one container only
> - effective container pointer (econtainer()) is used on the task to avoid
> insertion of additional argument "container" to all functions where it is
> required.
> - kernel compilation with disabled virtualization should result in old good
> linux kernel
>
> Patches following this one will be used for virtualization of the kernel
> resources based on this container infrastructure, including those VPID patches I
> sent before. Every virtualized resource can be given separate config option if
> needed (just give me to know if it is desired).

After having digested this I think there is something sane with regard to
this container idea. I still think the implementation is totally wrong but
there is a potential problem the basic idea solves.

In the traditional linux/plan9 style of namespaces the question is
which resources different tasks share, and we pass clone bits to determine
what we want to share and what we don't want to share. Semantically this
is very clean and allows for a great deal of flexibility. However
as the flexibility increases we get more code in do_fork more
reference counts and ultimately the performance decreases. In addition
it is not common for us to change which resources we share.

So our traditional route is flexible but it does not optimize the common
case where we share all of the same things. Containers can potentially
optimize that case. So long as anyone can create a new container even
someone inside a container we do not loose flexibility.

To deal with networking there are currently a significant number of
variables with static storage duration. Making those variables global

and placing them in structures is neither as efficient as it could be
nor is it as maintainable as it should be. Other subsystems have
similar problems.

So if we put econtainer in struct thread_info and optimize it like we
do current, create an interface to variables similar to
DEFINE_PER_CPU, create a syscall interface similar to clone, and
show that by doing this we don't loose flexibility, then it looks like
a good idea.

If we can't do better than the current clone model of shared resources
then this model feels like gratuitous change.

Eric

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Jeff Dike on Thu, 09 Feb 2006 02:18:28 GMT
View Forum Message <> Reply to Message

On Wed, Feb 08, 2006 at 05:24:16PM -0700, Eric W. Biederman wrote:
> To deal with networking there are currently a significant number of
> variables with static storage duration. Making those variables global
> and placing them in structures is neither as efficient as it could be
> nor is it as maintainable as it should be. Other subsystems have
> similar problems.

BTW, there is another solution, which you may or may not consider to
be clean.

That is to load a separate copy of the subsystem (code and data) as a
module when you want a new instance of it. The code doesn't change,
but you probably have to move it around some and provide some sort of
interface to it.

I did this to the scheduler last year - see
  http://marc.theaimsgroup.com/?l=linux-kernel&m=111404726 721747&w=2

   Jeff

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by ebiederm on Thu, 09 Feb 2006 03:16:38 GMT
View Forum Message <> Reply to Message

Jeff Dike <jdike@addtoit.com> writes:

> On Wed, Feb 08, 2006 at 05:24:16PM -0700, Eric W. Biederman wrote:
>> To deal with networking there are currently a significant number of
>> variables with static storage duration. Making those variables global
>> and placing them in structures is neither as efficient as it could be
>> nor is it as maintainable as it should be. Other subsystems have
>> similar problems.
>
> BTW, there is another solution, which you may or may not consider to
> be clean.
>
> That is to load a separate copy of the subsystem (code and data) as a
> module when you want a new instance of it. The code doesn't change,
> but you probably have to move it around some and provide some sort of
> interface to it.

There are some real drawbacks to that. Basically as things are structured
you only want multiple copies of the user facing data structures. Pretty
much everything else really does remain the same.

> I did this to the scheduler last year - see
> http://marc.theaimsgroup.com/?l=linux-kernel&m=111404726 721747&w=2

I will take a look but I don't expect to find anything.

Eric

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by Kyle Moffett on Thu, 09 Feb 2006 04:45:02 GMT
View Forum Message <> Reply to Message

On Feb 07, 2006, at 17:06, Eric W. Biederman wrote:
> I think I can boil the discussion down into some of the fundamental
> questions that we are facing.
>
> Currently everyone seems to agree that we need something like my
> namespace concept that isolates multiple resources.
>
> We need these for
> UIDS
> FILESYSTEM

I have one suggestion for this (it also covers capabilities to a
certain extent). Could we use the kernel credentials system to
abstract away the concept of a single UID/GID? We currently have
uid, euid, gid, egid, groups, fsid. I'm thinking that there would be
virtualized UID tables to determine ownership of processes/SHM/etc.

Each process would have a (uid_container,uid) pair (or similar) as
its "uid" and likewise for gid. Then the ability to send signals to
any given (uid_container,uid) or (gid_container,gid) pair would be
given by keys in the kernel keyring indexed by the "uid_container"
part and containing the "uid" part (or maybe just a pointer).

Likewise the filesystem access could be virtualized by using uid and
gid keys in the kernel keyring indexed by vfsmount (Not superblock,
so that it would be possible to have different UID representations on
different mounts/parts of the same filesystem).

I'm guessing that the performance implications of the above would not
be quite so nice, as it would put a lot of code in the fastpath, but
I would guess that it might be possible to use the existing fields
for processes without any virtualization needs.

Cheers,
Kyle Moffett

--
There is no way to make Linux robust with unreliable memory
subsystems, sorry. It would be like trying to make a human more
robust with an unreliable O2 supply. Memory just has to work.
  -- Andi Kleen

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by ebiederm on Thu, 09 Feb 2006 05:41:10 GMT
View Forum Message <> Reply to Message

Kyle Moffett <mrmacman_g4@mac.com> writes:

> On Feb 07, 2006, at 17:06, Eric W. Biederman wrote:
>> I think I can boil the discussion down into some of the fundamental questions
>> that we are facing.
>>
>> Currently everyone seems to agree that we need something like my namespace
>> concept that isolates multiple resources.
>>
>> We need these for
>> UIDS
>> FILESYSTEM
>
> I have one suggestion for this (it also covers capabilities to a certain
> extent). Could we use the kernel credentials system to abstract away the
> concept of a single UID/GID? We currently have uid, euid, gid, egid, groups,

> fsid. I'm thinking that there would be virtualized UID tables to determine
> ownership of processes/SHM/etc.
>
> Each process would have a (uid_container,uid) pair (or similar) as its "uid"
> and likewise for gid. Then the ability to send signals to any given
> (uid_container,uid) or (gid_container,gid) pair would be given by keys in the
> kernel keyring indexed by the "uid_container" part and containing the "uid"
> part (or maybe just a pointer).
>
> Likewise the filesystem access could be virtualized by using uid and gid keys
> in the kernel keyring indexed by vfsmount (Not superblock, so that it would be
> possible to have different UID representations on different mounts/parts of the
> same filesystem).
>
> I'm guessing that the performance implications of the above would not be quite
> so nice, as it would put a lot of code in the fastpath, but I would guess that
> it might be possible to use the existing fields for processes without any
> virtualization needs.

At least for signal sending it looks like it would be easier to just compare
the pointers to struct user. At least in that context it looks like it
would be as cheap as what we are doing now. I just don't know where
to find a struct user for the euid, or is it the normal uid.

Eric

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Hubertus Franke on Thu, 09 Feb 2006 16:38:31 GMT
View Forum Message <> Reply to Message

Jeff Dike wrote:
> On Wed, Feb 08, 2006 at 05:24:16PM -0700, Eric W. Biederman wrote:
>
>>To deal with networking there are currently a significant number of
>>variables with static storage duration. Making those variables global
>>and placing them in structures is neither as efficient as it could be
>>nor is it as maintainable as it should be. Other subsystems have
>>similar problems.
>
>
> BTW, there is another solution, which you may or may not consider to
> be clean.
>
> That is to load a separate copy of the subsystem (code and data) as a
> module when you want a new instance of it. The code doesn't change,
> but you probably have to move it around some and provide some sort of
> interface to it.

>
> I did this to the scheduler last year - see
> http://marc.theaimsgroup.com/?l=linux-kernel&m=111404726 721747&w=2
>
> Jeff
>

Jeff, interesting, but won't that post some serious scalability issue?
Imaging 100s of container/namespace ?

The namespace is mainly there to identify which data needs to be private
when multiple instances of a subsystem are considered and
encapsulate that data in an object/datastructure !

-- Hubertus

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Jeff Dike on Thu, 09 Feb 2006 17:47:58 GMT
View Forum Message <> Reply to Message

On Thu, Feb 09, 2006 at 11:38:31AM -0500, Hubertus Franke wrote:
> Jeff, interesting, but won't that post some serious scalability issue?
> Imaging 100s of container/namespace ?

In terms of memory?

Running size on sched.o gives me this on x86_64:
  text data bss dec hex filename
 35685 6880 28800 71365 116c5 sched.o

and on i386 (actually UML/i386)

  text data bss dec hex filename
 10010 36 2504 12550 3106 obj/kernel/sched.o

I'm not sure why there's such a big difference, but 100 instances adds
a meg or two (or three) to the kernel. This overstates things a bit
because there are things in sched.c which wouldn't be duplicated, like
the system calls.

How big a deal is that on a system which you plan to have 100s of
containers on anyway?

It's heavier than your namespaces, but does have the advantage that it
imposes no cost when it's not being used.

> The namespace is mainly there to identify which data needs to be private
> when multiple instances of a subsystem are considered and
> encapsulate that data in an object/datastructure !

Sure, and that's a fine approach. It's just not the only one.

    Jeff

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by ebiederm on Thu, 09 Feb 2006 21:56:46 GMT
View Forum Message <> Reply to Message

Ok digesting this some more I don't see a glaring case in the code
where it is set_econtainer(...); do stuff ; restore_econtainer(..);
has a tremendous advantage at the moment. Plus it removes the
opportunity to catch old code that deals with multiple contexts at
compile time, and has to be done by pain staking code review.

That aside the truly important thing I realized is that for
the optimization effects it is an implementation detail. Batched
reference counting of shared resources is something we can
go back in and add at any time. It will take a little refactoring
to do but it something that can be done without changing the
user space API.

So since individual namespace pointers in the task_struct
are simpler and have the required flexibility. I think we
can table this part of the discussion for now.

The other detail I brought up which was most interesting was
the marking of specific variables that would go into a
container or namespace. Well as long as it is implemented on
the namespace level that is still possible, and may even
be beneficial.

Eric

Subject: Re: [PATCH 1/4] Virtualization/containers: introduction
Posted by Sam Vilain on Thu, 09 Feb 2006 22:09:00 GMT
View Forum Message <> Reply to Message

Jeff Dike wrote:
> On Thu, Feb 09, 2006 at 11:38:31AM -0500, Hubertus Franke wrote:
>>Jeff, interesting, but won't that post some serious scalability issue?
>>Imaging 100s of container/namespace ?

> In terms of memory?
> Running size on sched.o gives me this on x86_64:
> text data bss dec hex filename
> 35685 6880 28800 71365 116c5 sched.o
>
> and on i386 (actually UML/i386)
>
> text data bss dec hex filename
> 10010 36 2504 12550 3106 obj/kernel/sched.o
>
> I'm not sure why there's such a big difference, but 100 instances adds
> a meg or two (or three) to the kernel. This overstates things a bit
> because there are things in sched.c which wouldn't be duplicated, like
> the system calls.
>
> How big a deal is that on a system which you plan to have 100s of
> containers on anyway?

Quite a big deal. You might have 2Gigs of main memory, but your CPU is unlikely to be more than a Megabyte in close reach. A meg or two of scheduler data and code means that your L1 and L2 cache will be cycling every scheduler round; which is OK if you have very short runqueues but as you get more and more processes it will really start to hurt.

Remember, systems today are memory bound and anything you can do to reduce the amount of time the system sits around waiting for memory to fetch, the better.

Compare that to the Token Bucket Scheduler of Linux-VServer; a tiny struct for each process umbrella, that will generally fit in one or two cachelines, to which the scheduling support adds four ints and a spinlock. With this it achieves fair CPU scheduling between vservers.

Sam.

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by ebiederm on Thu, 09 Feb 2006 22:25:35 GMT
View Forum Message <> Reply to Message

ebiederm@xmission.com (Eric W. Biederman) writes:

> I think I can boil the discussion down into some of the fundamental
> questions that we are facing.
>
> Currently everyone seems to agree that we need something like
> my namespace concept that isolates multiple resources.

>
> We need these for
> PIDS
> UIDS
> SYSVIPC
> NETWORK
> UTSNAME
> FILESYSTEM
> etc.
>
> The questions seem to break down into:
> 1) Where do we put the references to the different namespaces?
> - Do we put the references in a struct container that we reference from
> struct task_struct?
> - Do we put the references directly in struct task_struct?

Answer in the task_struct. It is the simplest and most flexible
route and the other implementations are still possible.

> 2) What is the syscall interface to create these namespaces?
> - Do we add clone flags?
> (Plan 9 style)
> - Do we add a syscall (similar to setsid) per namespace?
> (Traditional unix style)?
> - Do we in addition add syscalls to manipulate containers generically?

The answer seems to be we decide on a per namespace basis with
additional syscalls being mandatory if we have any additional data to
pass.

> 3) How do we refer to namespaces and containers when we are not members?

I have seen no arguments against referring to namespaces or containers
by global ids. So it seems we do not need a container id.

> 4) How do we implement each of these namespaces?
> Besides being maintainable are there other constraints?

Largely quite. But I have not heard additional constraints.

> 5) How do we control the resource inside a namespace starting
> from a process that is outside of that namespace?
> - The filesystem mount namespace gave an interesting answer.
> So it is quite possible other namespaces will give
> equally interesting and surprising answers.

Not yet resolved, but a bit of speculation.

> 6) How do we do all of this efficiently without a noticeable impact on
>   performance?
>   - I have already heard concerns that I might be introducing cache
>   line bounces and thus increasing tasklist_lock hold time.
>   Which on big way systems can be a problem.

A little discussion. At the level of the last few cache line I
think this needs to be addressed when we merge. Simply not
messing up existing optimizations sounds like a good initial
target. Basically at this stage trying hard would be a
premature optimization.

> 7) How do we allow a process inside a container to create containers
>   for it's children?
>   - In general this is trivial but there are a few ugly issues
>   here.

This look mostly like something to be discussed when we merge
namespaces. But as long as we keep it in mind it is easy.

Eric

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by dev on Mon, 20 Feb 2006 12:08:58 GMT
View Forum Message <> Reply to Message

>> The questions seem to break down into:
>> 1) Where do we put the references to the different namespaces?
>>   - Do we put the references in a struct container that we reference
>> from struct task_struct?
>>   - Do we put the references directly in struct task_struct?
>
>
> You "cache"   task_struct->container->hotsubsys   under
> task_struct->hotsubsys.
> We don't change containers other then at clone time, so no coherency
> issue here !!!!
> Which subsystems pointers to "cache", should be agreed by the experts,
> but first approach should always not to cache and go through the container.
agreed. I see no much reason to cache it and make tons of the same
pointers in all the tasks. Only if needed.
Also, in OpenVZ container has many fields intergrated inside, so there
is no additional dereference, but task->container->subsys_field

>> 2) What is the syscall interface to create these namespaces?
>>   - Do we add clone flags?   (Plan 9 style)

> Like that approach .. flexible .. particular when one has well specified
> namespaces.
mmm, how do you plan to pass additional flags to clone()?
e.g. strong or weak isolation of pids?

another questions:
how do you plan to meet the dependancies between namespaces?
e.g. conntracks require netfilters to be initialized.
network requires sysctls and proc to be initialized and so on.
do you propose to track all this in clone()? huh...

>> - Do we add a syscall (similar to setsid) per namespace?
>> (Traditional unix style)?
can be so...

>> - Do we in addition add syscalls to manipulate containers generically?
>>
>> I don't think having a single system call to create a container and
>> a new
>> instance of each namespace is reasonable as that does not give us a
>> path into the future when we create yet another namespace.
>>
> Agreed.
why do you think so?
this syscalls will start handling this new namespace and that's all.
this is not different from many syscalls approach.

>> 4) How do we implement each of these namespaces?
>> Besides being maintainable are there other constraints?
>>
> Good question... at least with PID and FS two are there ..
>>
>> 6) How do we do all of this efficiently without a noticeable impact on
>> performance?
>> - I have already heard concerns that I might be introducing cache
>> line bounces and thus increasing tasklist_lock hold time.
>> Which on big way systems can be a problem.
this is nothing compared to hierarchy operations.
BTW, heirarchy also introduces complicated resource accounting,
sometimes making it even impossible.

Kirill

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by Herbert Poetzl on Mon, 20 Feb 2006 12:40:55 GMT

On Mon, Feb 20, 2006 at 03:11:32PM +0300, Kirill Korotaev wrote:
> >>The questions seem to break down into:
> >>1) Where do we put the references to the different namespaces?
> >> - Do we put the references in a struct container that we reference
> >>from struct task_struct?
> >> - Do we put the references directly in struct task_struct?
> >
> >
> >You "cache" task_struct->container->hotsubsys under
> >task_struct->hotsubsys.
> >We don't change containers other then at clone time, so no coherency
> >issue here !!!!
> >Which subsystems pointers to "cache", should be agreed by the experts,
> >but first approach should always not to cache and go through the container.
> agreed. I see no much reason to cache it and make tons of the same
> pointers in all the tasks. Only if needed.

> Also, in OpenVZ container has many fields intergrated inside, so there
> is no additional dereference, but task->container->subsys_field

as does Linux-VServer currently, but do you have
any proof that putting all the fields together in
one big structure actually has any (dis)advantage
over separate structures?

> >>2) What is the syscall interface to create these namespaces?
> >> - Do we add clone flags? (Plan 9 style)
> >Like that approach .. flexible .. particular when one has well
> >specified namespaces.
> mmm, how do you plan to pass additional flags to clone()?
> e.g. strong or weak isolation of pids?

do you really have to pass them at clone() time?
would shortly after be more than enough?
what if you want to change those properties later?

> another questions:
> how do you plan to meet the dependancies between namespaces?
> e.g. conntracks require netfilters to be initialized.
> network requires sysctls and proc to be initialized and so on.
> do you propose to track all this in clone()? huh...

this is missing isolation/virtualization, and I guess
it has to be done to make those spaces useful ...

> >> - Do we add a syscall (similar to setsid) per namespace?
> >> (Traditional unix style)?

> can be so...
>
> >> - Do we in addition add syscalls to manipulate containers
> >> generically?
> >>
> >> I don't think having a single system call to create a container
> >> and a new instance of each namespace is reasonable as that does
> >> not give us a path into the future when we create yet another
> >> namespace.
> >Agreed.
> why do you think so?

> this syscalls will start handling this new namespace and that's all.
> this is not different from many syscalls approach.

well, let's defer the 'how amny syscalls' issue to
a later time, when we know what we want to implement :)

> >>4) How do we implement each of these namespaces?
> >> Besides being maintainable are there other constraints?
> >>
> >Good question... at least with PID and FS two are there ..
> >>
> >>6) How do we do all of this efficiently without a noticeable impact on
> >> performance?
> >> - I have already heard concerns that I might be introducing cache
> >> line bounces and thus increasing tasklist_lock hold time.
> >> Which on big way systems can be a problem.

> this is nothing compared to hierarchy operations.
> BTW, heirarchy also introduces complicated resource accounting,
> sometimes making it even impossible.

well, depends how you do it ...

best,
Herbert

> Kirill

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by dev on Mon, 20 Feb 2006 14:25:35 GMT
View Forum Message <> Reply to Message

> as does Linux-VServer currently, but do you have
> any proof that putting all the fields together in

> one big structure actually has any (dis)advantage
> over separate structures?
have no proof and don't mind if there are many pointers. Though this
doesn't look helpful to me as well.

>>mmm, how do you plan to pass additional flags to clone()?
>>e.g. strong or weak isolation of pids?
> do you really have to pass them at clone() time?
> would shortly after be more than enough?
> what if you want to change those properties later?
I don't think it is always suiatable to do configuration later.
We had races in OpenVZ on VPS create/stop against exec/enter etc. (even
introduced flag is_running). So I have some experience to believe it
will be painfull place.

>>this syscalls will start handling this new namespace and that's all.
>>this is not different from many syscalls approach.
> well, let's defer the 'how amny syscalls' issue to
> a later time, when we know what we want to implement :)
agreed.

Kirill

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by Herbert Poetzl on Mon, 20 Feb 2006 15:16:14 GMT
View Forum Message <> Reply to Message

On Mon, Feb 20, 2006 at 05:26:13PM +0300, Kirill Korotaev wrote:
>> as does Linux-VServer currently, but do you have
>> any proof that putting all the fields together in
>> one big structure actually has any (dis)advantage
>> over separate structures?

> have no proof and don't mind if there are many pointers.
> Though this doesn't look helpful to me as well.

well, my point is just that we don't know yet
so we should not favor one over the other, just
because somebody did it like that and it didn't
hurt :)

>>> mmm, how do you plan to pass additional flags to clone()?
>>> e.g. strong or weak isolation of pids?

>> do you really have to pass them at clone() time?
>> would shortly after be more than enough?

>> what if you want to change those properties later?

> I don't think it is always suiatable to do configuration later.
> We had races in OpenVZ on VPS create/stop against exec/enter etc.
> (even introduced flag is_running).
> So I have some experience to believe it will be painfull place.

well, Linux-VServer uses a state called 'setup'
which allows to change all kinds of things before
the guest can be entered, this state is changed
as the last operation of the setup, which in turn
drops all the capabilities and makes the guest
visible to the outside ...

works quite well and seems to be free of those
races you mentioned ...

>>> this syscalls will start handling this new namespace and that's all.
>>> this is not different from many syscalls approach.
>> well, let's defer the 'how amny syscalls' issue to
>> a later time, when we know what we want to implement :)
> agreed.

btw, maybe it's just me, but would it be possible
to do the email quoting like this:

>>> Text

instead of

> >>Text

TIA,
Herbert

> Kirill

Subject: Re: The issues for agreeing on a virtualization/namespaces implementation.
Posted by Andi Kleen on Mon, 20 Feb 2006 15:17:51 GMT
View Forum Message <> Reply to Message

If you guys want to continue to argue this point forever please drop
me from cc.

Thanks.

-Andi