Subject: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Kirill Korotaev on Fri, 03 Feb 2006 16:57:18 GMT
View Forum Message <> Reply to Message

This patch introduces some abstract container/VPS kernel structure and
tiny amount of operations on it.

Patches following this one will be used for virtualization of some
resources based on this container infrastructure, including those VPIDs
patches I sent before.

What is important here is that:
- each VPS has unique ID
- each process in kernel can belong to one VPS only

Kirill

```
--- ./include/linux/vps_info.h.vps_info 2006-02-03 16:38:33.000000000 +0300
+++ ./include/linux/vps_info.h 2006-02-03 16:49:26.000000000 +0300
@@ -0,0 +1,41 @@
+#ifndef __LINUX_VPS_INFO_H_
+#define __LINUX_VPS_INFO_H_
+
+#include <asm/types.h>
+#include <asm/atomic.h>
+
+struct task_struct;
+
+struct vps_info {
+ u32 id;
+ struct task_struct *init_task;
+ atomic_t refcnt;
+};
+
+extern struct vps_info host_vps_info;
+typedef struct vps_info *vps_t;
+
+static inline vps_t get_vps(vps_t vps)
+{
+ atomic_inc(&vps->refcnt);
+ return vps;
+}
+
+static inline void put_vps(vps_t vps)
+{
+ atomic_dec(&vps->refcnt);
+}
+
```

```
+#include <linux/sched.h>
+#include <asm/current.h>
+
+static inline vps_t set_current_vps(vps_t vps)
+{
+ vps_t ret;
+
+ ret = current->owner_vps;
+ current->owner_vps = vps;
+ return ret;
+}
+
+#endif
--- ./include/linux/sched.h.vps_info 2006-02-03 16:38:33.000000000 +0300
+++ ./include/linux/sched.h 2006-02-03 16:43:42.000000000 +0300
@@ -687,6 +687,7 @@

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

 struct task_struct {
  volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
@@ -845,6 +846,7 @@
  struct backing_dev_info *backing_dev_info;

  struct io_context *io_context;
+ struct vps_info *owner_vps;

  unsigned long ptrace_message;
  siginfo_t *last_siginfo; /* For ptrace use.  */
@@ -875,6 +877,8 @@
  struct rcu_head rcu;
 };

+#define current_vps() (current->owner_vps)
+
 static inline pid_t process_group(struct task_struct *tsk)
 {
  return tsk->signal->pgrp;
--- ./include/linux/init_task.h.vps_info 2006-02-03 16:38:33.000000000 +0300
+++ ./include/linux/init_task.h 2006-02-03 16:43:18.000000000 +0300
@@ -3,6 +3,7 @@

 #include <linux/file.h>
 #include <linux/rcupdate.h>
+#include <linux/vps_info.h>
```

```
 #define INIT_FDTABLE \
 {      \
@@ -121,6 +122,7 @@
  .journal_info = NULL,      \
  .cpu_timers = INIT_CPU_TIMERS(tsk.cpu_timers),  \
  .fs_excl = ATOMIC_INIT(0),    \
+ .owner_vps = &host_vps_info,    \
 }


--- ./init/main.c.vps_info 2006-02-03 16:38:33.000000000 +0300
+++ ./init/main.c 2006-02-03 16:43:18.000000000 +0300
@@ -47,6 +47,7 @@
 #include <linux/rmap.h>
 #include <linux/mempolicy.h>
 #include <linux/key.h>
+#include <linux/vps_info.h>

 #include <asm/io.h>
 #include <asm/bugs.h>
@@ -434,6 +435,13 @@
  done = 1;
 }

+struct vps_info host_vps_info = {
+ .id  = 0,
+ .init_task = &init_task,
+ .refcnt  = ATOMIC_INIT(1),
+};
+
+EXPORT_SYMBOL(host_vps_info);
 /*
  * Activate the first processor.
  */
--- ./kernel/fork.c.vps_info 2006-02-03 16:38:33.000000000 +0300
+++ ./kernel/fork.c 2006-02-03 16:43:18.000000000 +0300
@@ -44,6 +44,7 @@
 #include <linux/rmap.h>
 #include <linux/acct.h>
 #include <linux/cn_proc.h>
+#include <linux/vps_info.h>

 #include <asm/pgtable.h>
 #include <asm/pgalloc.h>
@@ -1132,6 +1133,7 @@
  p->ioprio = current->ioprio;

  SET_LINKS(p);
```

```
+ get_vps(p->owner_vps);
  if (unlikely(p->ptrace & PT_PTRACED))
   __ptrace_link(p, current->parent);

--- ./kernel/exit.c.vps_info 2006-02-03 16:38:33.000000000 +0300
+++ ./kernel/exit.c 2006-02-03 16:43:18.000000000 +0300
@@ -31,6 +31,7 @@
 #include <linux/signal.h>
 #include <linux/cn_proc.h>
 #include <linux/mutex.h>
+#include <linux/vps_info.h>

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

  p = leader;
```

Subject: [RFC][PATCH 2/5] Virtualization/containers: UIDs
Posted by dev on Fri, 03 Feb 2006 17:01:30 GMT

View Forum Message <> Reply to Message

The simplest virtualization of UID hashes, just to demonstrate our approach. Each container has it's own set of uids and should simply allocate hash/initialize it on startup.

Kirill

```
--- ./include/linux/vps_info.h.vps_uid_hash 2006-02-03 16:49:26.000000000 +0300
+++ ./include/linux/vps_info.h 2006-02-03 16:49:51.000000000 +0300
@@ -5,11 +5,14 @@
 #include <asm/atomic.h>

 struct task_struct;
+struct list_head;

 struct vps_info {
  u32 id;
  struct task_struct *init_task;
  atomic_t refcnt;
+
+ struct list_head *vps_uid_hash;
```

```
 };

 extern struct vps_info host_vps_info;
--- ./kernel/user.c.vps_uid_hash 2006-02-03 16:49:08.000000000 +0300
+++ ./kernel/user.c 2006-02-03 16:49:51.000000000 +0300
@@ -14,6 +14,7 @@
 #include <linux/bitops.h>
 #include <linux/key.h>
 #include <linux/interrupt.h>
+#include <linux/vps_info.h>

 /*
  * UID task count cache, to get fast user lookup in "alloc_uid"
@@ -24,7 +25,8 @@
 #define UIDHASH_SZ  (1 << UIDHASH_BITS)
 #define UIDHASH_MASK  (UIDHASH_SZ - 1)
 #define __uidhashfn(uid) (((uid >> UIDHASH_BITS) + uid) & UIDHASH_MASK)
-#define uidhashentry(uid) (uidhash_table + __uidhashfn((uid)))
+#define uidhashentry(uid) (current_vps()->vps_uid_hash + \
+      __uidhashfn((uid)))

 static kmem_cache_t *uid_cachep;
 static struct list_head uidhash_table[UIDHASH_SZ];
@@ -200,6 +202,7 @@

  /* Insert the root user immediately (init already runs as root) */
  spin_lock_irq(&uidhash_lock);
+ host_vps_info.vps_uid_hash = uidhash_table;
  uid_hash_insert(&root_user, uidhashentry(0));
  spin_unlock_irq(&uidhash_lock);
```

Subject: [RFC][PATCH 3/5] Virtualization/containers: UTSNAME
Posted by dev on Fri, 03 Feb 2006 17:04:50 GMT
View Forum Message <> Reply to Message

Virtualization of UTSNAME.
As simple as UID hashes, just virtualizes system_utsname variable
(diff-vps-uts-name-core) and replaces access to it with vps_utsname in
required places (diff-vps-uts-name-kern).

Kirill

```
--- ./include/linux/vps_info.h.vps_uts_core 2006-02-03 16:49:51.000000000 +0300
+++ ./include/linux/vps_info.h 2006-02-03 16:50:39.000000000 +0300
@@ -6,6 +6,7 @@

 struct task_struct;
```

```
 struct list_head;
+struct new_utsname;

 struct vps_info {
  u32 id;
@@ -13,6 +14,7 @@
  atomic_t refcnt;

  struct list_head *vps_uid_hash;
+ struct new_utsname *vps_uts_name;
 };

 extern struct vps_info host_vps_info;
--- ./include/linux/utsname.h.vps_uts_core 2006-02-03 16:38:17.000000000 +0300
+++ ./include/linux/utsname.h 2006-02-03 16:50:26.000000000 +0300
@@ -1,6 +1,8 @@
 #ifndef _LINUX_UTSNAME_H
 #define _LINUX_UTSNAME_H

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

 struct oldold_utsname {
@@ -31,6 +33,7 @@
 };

 extern struct new_utsname system_utsname;
+#define vps_utsname (*(current_vps()->vps_uts_name))

 extern struct rw_semaphore uts_sem;
 #endif
--- ./init/main.c.vps_uts_core 2006-02-03 16:43:18.000000000 +0300
+++ ./init/main.c 2006-02-03 16:50:26.000000000 +0300
@@ -439,6 +439,7 @@
  .id  = 0,
  .init_task = &init_task,
  .refcnt  = ATOMIC_INIT(1),
+ .vps_uts_name = &system_utsname,
 };

 EXPORT_SYMBOL(host_vps_info);

--- ./arch/alpha/kernel/osf_sys.c.vps_uts 2006-02-03 11:56:22.000000000 +0300
+++ ./arch/alpha/kernel/osf_sys.c 2006-02-03 13:35:25.000000000 +0300
@@ -402,15 +402,15 @@

  down_read(&uts_sem);
```

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

  error = 0;
@@ -449,8 +449,8 @@

  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(vps_utsname.domainname[i], name + i);
+  if (vps_utsname.domainname[i] == '\0')
     break;
  }
  up_read(&uts_sem);
@@ -608,11 +608,11 @@
 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,
+  vps_utsname.sysname,
+  vps_utsname.nodename,
+  vps_utsname.release,
+  vps_utsname.version,
+  vps_utsname.machine,
    "alpha", /* instruction set architecture */
    "dummy", /* hardware serial number */
    "dummy", /* hardware manufacturer */
--- ./arch/i386/kernel/sys_i386.c.vps_uts 2006-01-03 06:21:10.000000000 +0300
+++ ./arch/i386/kernel/sys_i386.c 2006-02-03 13:35:25.000000000 +0300
```

```
@@ -217,7 +217,7 @@
  if (!name)
   return -EFAULT;
  down_read(&uts_sem);
- err=copy_to_user(name, &system_utsname, sizeof (*name));
+ err=copy_to_user(name, &vps_utsname, sizeof (*name));
  up_read(&uts_sem);
  return err?-EFAULT:0;
 }
@@ -233,15 +233,15 @@

    down_read(&uts_sem);

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

  up_read(&uts_sem);
--- ./arch/m32r/kernel/sys_m32r.c.vps_uts 2006-01-03 06:21:10.000000000 +0300
+++ ./arch/m32r/kernel/sys_m32r.c 2006-02-03 13:35:25.000000000 +0300
@@ -199,7 +199,7 @@
  if (!name)
   return -EFAULT;
  down_read(&uts_sem);
- err=copy_to_user(name, &system_utsname, sizeof (*name));
+ err=copy_to_user(name, &vps_utsname, sizeof (*name));
  up_read(&uts_sem);
  return err?-EFAULT:0;
 }
--- ./arch/mips/kernel/linux32.c.vps_uts 2006-02-03 11:56:03.000000000 +0300
+++ ./arch/mips/kernel/linux32.c 2006-02-03 13:35:25.000000000 +0300
@@ -1150,7 +1150,7 @@
  int ret = 0;

  down_read(&uts_sem);
- if (copy_to_user(name,&system_utsname,sizeof *name))
```

```
+ if (copy_to_user(name,&vps_utsname,sizeof *name))
   ret = -EFAULT;
  up_read(&uts_sem);

--- ./arch/mips/kernel/syscall.c.vps_uts 2006-02-03 11:56:03.000000000 +0300
+++ ./arch/mips/kernel/syscall.c 2006-02-03 13:35:25.000000000 +0300
@@ -229,7 +229,7 @@
  */
 asmlinkage int sys_uname(struct old_utsname * name)
 {
- if (name && !copy_to_user(name, &system_utsname, sizeof (*name)))
+ if (name && !copy_to_user(name, &vps_utsname, sizeof (*name)))
   return 0;
  return -EFAULT;
 }
@@ -246,15 +246,15 @@
  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,&vps_utsname.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,&vps_utsname.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,&vps_utsname.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,&vps_utsname.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,&vps_utsname.machine,__OLD_UTS_LEN);
  error = __put_user(0,name->machine+__OLD_UTS_LEN);
  error = error ? -EFAULT : 0;

@@ -290,10 +290,10 @@
    return -EFAULT;

   down_write(&uts_sem);
-  strncpy(system_utsname.nodename, nodename, len);
+  strncpy(vps_utsname.nodename, nodename, len);
   nodename[__NEW_UTS_LEN] = '\0';
-  strlcpy(system_utsname.nodename, nodename,
-          sizeof(system_utsname.nodename));
+  strlcpy(vps_utsname.nodename, nodename,
+          sizeof(vps_utsname.nodename));
   up_write(&uts_sem);
```

```
   return 0;
  }
--- ./arch/mips/kernel/sysirix.c.vps_uts 2006-02-03 11:56:03.000000000 +0300
+++ ./arch/mips/kernel/sysirix.c 2006-02-03 13:35:25.000000000 +0300
@@ -904,7 +904,7 @@
  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, vps_utsname.domainname, len) ? -EFAULT : 0;
  up_read(&uts_sem);

  return err;
@@ -1147,11 +1147,11 @@
 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(vps_utsname.sysname, buf->sysname, 65)
+     || copy_from_user(vps_utsname.nodename, buf->nodename, 65)
+     || copy_from_user(vps_utsname.release, buf->release, 65)
+     || copy_from_user(vps_utsname.version, buf->version, 65)
+     || copy_from_user(vps_utsname.machine, buf->machine, 65)) {
   return -EFAULT;
  }
  up_read(&uts_sem);
--- ./arch/parisc/hpux/sys_hpux.c.vps_uts 2006-02-03 11:56:03.000000000 +0300
+++ ./arch/parisc/hpux/sys_hpux.c 2006-02-03 13:35:25.000000000 +0300
@@ -266,15 +266,15 @@

  down_read(&uts_sem);

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

  up_read(&uts_sem);
@@ -373,8 +373,8 @@
   /*  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(vps_utsname.sysname, ubuf, len)) {
+   vps_utsname.sysname[len] = 0;
    error = 0;
   }
   up_write(&uts_sem);
@@ -400,8 +400,8 @@
   /*  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(vps_utsname.release, ubuf, len)) {
+   vps_utsname.release[len] = 0;
    error = 0;
   }
   up_write(&uts_sem);
@@ -422,13 +422,13 @@

   down_read(&uts_sem);

- nlen = strlen(system_utsname.domainname) + 1;
+ nlen = strlen(vps_utsname.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, vps_utsname.domainname, len))
   goto done;
  err = 0;
 done:
--- ./arch/powerpc/kernel/syscalls.c.vps_uts 2006-02-03 11:56:03.000000000 +0300
+++ ./arch/powerpc/kernel/syscalls.c 2006-02-03 13:35:25.000000000 +0300
@@ -259,7 +259,7 @@
  int err = 0;
```

```
  down_read(&uts_sem);
- if (copy_to_user(name, &system_utsname, sizeof(*name)))
+ if (copy_to_user(name, &vps_utsname, sizeof(*name)))
    err = -EFAULT;
  up_read(&uts_sem);
  if (!err)
@@ -272,7 +272,7 @@
  int err = 0;

  down_read(&uts_sem);
- if (copy_to_user(name, &system_utsname, sizeof(*name)))
+ if (copy_to_user(name, &vps_utsname, sizeof(*name)))
    err = -EFAULT;
  up_read(&uts_sem);
  if (!err)
@@ -288,19 +288,19 @@
    return -EFAULT;

  down_read(&uts_sem);
- error = __copy_to_user(&name->sysname, &system_utsname.sysname,
+ error = __copy_to_user(&name->sysname, &vps_utsname.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, &vps_utsname.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, &vps_utsname.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, &vps_utsname.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, &vps_utsname.machine,
      __OLD_UTS_LEN);
  error |= override_machine(name->machine);
  up_read(&uts_sem);
--- ./arch/sh/kernel/setup.c.vps_uts 2006-02-03 11:56:22.000000000 +0300
+++ ./arch/sh/kernel/setup.c 2006-02-03 13:35:25.000000000 +0300
@@ -485,7 +485,7 @@
   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", vps_utsname.machine);
```

```
  seq_printf(m, "cpu type\t: %s\n", get_cpu_subtype());

  show_cpuflags(m);
--- ./arch/sh/kernel/sys_sh.c.vps_uts 2006-01-03 06:21:10.000000000 +0300
+++ ./arch/sh/kernel/sys_sh.c 2006-02-03 13:35:25.000000000 +0300
@@ -267,7 +267,7 @@
  if (!name)
   return -EFAULT;
  down_read(&uts_sem);
- err=copy_to_user(name, &system_utsname, sizeof (*name));
+ err=copy_to_user(name, &vps_utsname, sizeof (*name));
  up_read(&uts_sem);
  return err?-EFAULT:0;
 }
--- ./arch/sh64/kernel/sys_sh64.c.vps_uts 2006-01-03 06:21:10.000000000 +0300
+++ ./arch/sh64/kernel/sys_sh64.c 2006-02-03 13:35:25.000000000 +0300
@@ -279,7 +279,7 @@
  if (!name)
   return -EFAULT;
  down_read(&uts_sem);
- err=copy_to_user(name, &system_utsname, sizeof (*name));
+ err=copy_to_user(name, &vps_utsname, sizeof (*name));
  up_read(&uts_sem);
  return err?-EFAULT:0;
 }
--- ./arch/sparc/kernel/sys_sparc.c.vps_uts 2006-01-03 06:21:10.000000000 +0300
+++ ./arch/sparc/kernel/sys_sparc.c 2006-02-03 13:35:25.000000000 +0300
@@ -470,13 +470,13 @@

   down_read(&uts_sem);

- nlen = strlen(system_utsname.domainname) + 1;
+ nlen = strlen(vps_utsname.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, vps_utsname.domainname, len))
   goto done;
  err = 0;
 done:
--- ./arch/sparc/kernel/sys_sunos.c.vps_uts 2006-02-03 11:56:03.000000000 +0300
+++ ./arch/sparc/kernel/sys_sunos.c 2006-02-03 13:35:25.000000000 +0300
@@ -483,13 +483,13 @@
 {
  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], &vps_utsname.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], &vps_utsname.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], &vps_utsname.release[0], sizeof(name->rel) - 1);
+  ret |= __copy_to_user(&name->ver[0], &vps_utsname.version[0], sizeof(name->ver) - 1);
+  ret |= __copy_to_user(&name->mach[0], &vps_utsname.machine[0], sizeof(name->mach) - 1);
  }
  up_read(&uts_sem);
  return ret ? -EFAULT : 0;
--- ./arch/sparc64/kernel/sys_sparc.c.vps_uts 2006-01-03 06:21:10.000000000 +0300
+++ ./arch/sparc64/kernel/sys_sparc.c 2006-02-03 13:35:25.000000000 +0300
@@ -476,13 +476,13 @@

   down_read(&uts_sem);

- nlen = strlen(system_utsname.domainname) + 1;
+ nlen = strlen(vps_utsname.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, vps_utsname.domainname, len))
   goto done;
  err = 0;
 done:
--- ./arch/sparc64/kernel/sys_sunos32.c.vps_uts 2006-02-03 11:56:03.000000000 +0300
+++ ./arch/sparc64/kernel/sys_sunos32.c 2006-02-03 13:35:25.000000000 +0300
@@ -439,16 +439,16 @@
  int ret;

  down_read(&uts_sem);
- ret = copy_to_user(&name->sname[0], &system_utsname.sysname[0],
+ ret = copy_to_user(&name->sname[0], &vps_utsname.sysname[0],
       sizeof(name->sname) - 1);
- ret |= copy_to_user(&name->nname[0], &system_utsname.nodename[0],
```

```
+ ret |= copy_to_user(&name->nname[0], &vps_utsname.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], &vps_utsname.release[0],
        sizeof(name->rel) - 1);
- ret |= copy_to_user(&name->ver[0], &system_utsname.version[0],
+ ret |= copy_to_user(&name->ver[0], &vps_utsname.version[0],
        sizeof(name->ver) - 1);
- ret |= copy_to_user(&name->mach[0], &system_utsname.machine[0],
+ ret |= copy_to_user(&name->mach[0], &vps_utsname.machine[0],
        sizeof(name->mach) - 1);
  up_read(&uts_sem);
  return (ret ? -EFAULT : 0);
--- ./arch/sparc64/solaris/misc.c.vps_uts 2006-01-03 06:21:10.000000000 +0300
+++ ./arch/sparc64/solaris/misc.c 2006-02-03 13:35:25.000000000 +0300
@@ -239,7 +239,7 @@
   /* 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, vps_utsname.nodename,
         1, 1);
   up_read(&uts_sem);
   err |= set_utsfield(v->release, "2.6", 0, 0);
@@ -263,7 +263,7 @@
  /* 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, vps_utsname.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 @@
  case SI_HOSTNAME:
   r = buffer + 256;
   down_read(&uts_sem);
-  for (p = system_utsname.nodename, q = buffer;
+  for (p = vps_utsname.nodename, q = buffer;
        q < r && *p && *p != '.'; *q++ = *p++);
   up_read(&uts_sem);
   *q = 0;
--- ./arch/um/kernel/syscall_kern.c.vps_uts 2006-01-03 06:21:10.000000000 +0300
+++ ./arch/um/kernel/syscall_kern.c 2006-02-03 13:35:25.000000000 +0300
@@ -110,7 +110,7 @@
  if (!name)
   return -EFAULT;
```

```
  down_read(&uts_sem);
- err=copy_to_user(name, &system_utsname, sizeof (*name));
+ err=copy_to_user(name, &vps_utsname, sizeof (*name));
  up_read(&uts_sem);
  return err?-EFAULT:0;
 }
@@ -126,19 +126,19 @@

    down_read(&uts_sem);

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

--- ./arch/um/sys-x86_64/syscalls.c.vps_uts 2006-01-03 06:21:10.000000000 +0300
+++ ./arch/um/sys-x86_64/syscalls.c 2006-02-03 13:35:25.000000000 +0300
@@ -21,7 +21,7 @@
 {
  int err;
  down_read(&uts_sem);
- err = copy_to_user(name, &system_utsname, sizeof (*name));
+ err = copy_to_user(name, &vps_utsname, 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.vps_uts 2006-02-03 11:56:04.000000000 +0300
+++ ./arch/x86_64/ia32/sys_ia32.c 2006-02-03 13:35:25.000000000 +0300
@@ -868,13 +868,13 @@

    down_read(&uts_sem);
```

```
- error = __copy_to_user(&name->sysname,&system_utsname.sysname,__OLD_UTS_LEN);
+ error = __copy_to_user(&name->sysname,&vps_utsname.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,&vps_utsname.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,&vps_utsname.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,&vps_utsname.version,__OLD_UTS_LEN);
   __put_user(0,name->version+__OLD_UTS_LEN);
   {
    char *arch = "x86_64";
@@ -897,7 +897,7 @@
  if (!name)
   return -EFAULT;
  down_read(&uts_sem);
- err=copy_to_user(name, &system_utsname, sizeof (*name));
+ err=copy_to_user(name, &vps_utsname, 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.vps_uts 2006-01-03 06:21:10.000000000 +0300
+++ ./arch/x86_64/kernel/sys_x86_64.c 2006-02-03 13:35:25.000000000 +0300
@@ -148,7 +148,7 @@
 {
  int err;
  down_read(&uts_sem);
- err = copy_to_user(name, &system_utsname, sizeof (*name));
+ err = copy_to_user(name, &vps_utsname, 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.vps_uts 2006-01-03 06:21:10.000000000 +0300
+++ ./arch/xtensa/kernel/syscalls.c 2006-02-03 13:35:25.000000000 +0300
@@ -129,7 +129,7 @@

 int sys_uname(struct old_utsname * name)
 {
- if (name && !copy_to_user(name, &system_utsname, sizeof (*name)))
+ if (name && !copy_to_user(name, &vps_utsname, sizeof (*name)))
   return 0;
  return -EFAULT;
 }
--- ./include/asm-i386/elf.h.vps_uts 2006-01-03 06:21:10.000000000 +0300
+++ ./include/asm-i386/elf.h 2006-02-03 13:35:25.000000000 +0300
@@ -108,7 +108,7 @@
```

```
    For the moment, we have only optimizations for the Intel generations,
    but that could change... */

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

 #ifdef __KERNEL__
 #define SET_PERSONALITY(ex, ibcs2) do { } while (0)
--- ./kernel/sys.c.vps_uts 2006-02-03 11:56:24.000000000 +0300
+++ ./kernel/sys.c 2006-02-03 13:35:25.000000000 +0300
@@ -1518,7 +1518,7 @@
  int errno = 0;

  down_read(&uts_sem);
- if (copy_to_user(name,&system_utsname,sizeof *name))
+ if (copy_to_user(name,&vps_utsname,sizeof *name))
   errno = -EFAULT;
  up_read(&uts_sem);
  return errno;
@@ -1536,8 +1536,8 @@
  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(vps_utsname.nodename, tmp, len);
+  vps_utsname.nodename[len] = 0;
   errno = 0;
  }
  up_write(&uts_sem);
@@ -1553,11 +1553,11 @@
  if (len < 0)
   return -EINVAL;
  down_read(&uts_sem);
- i = 1 + strlen(system_utsname.nodename);
+ i = 1 + strlen(vps_utsname.nodename);
  if (i > len)
   i = len;
  errno = 0;
- if (copy_to_user(name, system_utsname.nodename, i))
+ if (copy_to_user(name, vps_utsname.nodename, i))
   errno = -EFAULT;
  up_read(&uts_sem);
  return errno;
@@ -1582,8 +1582,8 @@
  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(vps_utsname.domainname, tmp, len);
+  vps_utsname.domainname[len] = 0;
   errno = 0;
  }
  up_write(&uts_sem);
```

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Linus Torvalds on Fri, 03 Feb 2006 17:15:19 GMT
View Forum Message <> Reply to Message

On Fri, 3 Feb 2006, Kirill Korotaev wrote:
>
> This patch introduces some abstract container/VPS kernel structure and tiny
> amount of operations on it.

Please don't use things like "vps_t".

It's a _mistake_ to use typedef for structures and pointers. When you see
a

```
 vps_t a;
```

in the source, what does it mean?

In contrast, if it says

```
 struct virtual_container *a;
```

you can actually tell what "a" is.

Lots of people think that typedefs "help readability". Not so. They are
useful only for

 (a) totally opaque objects (where the typedef is actively used to _hide_
     what the object is).

     Example: "pte_t" etc opaque objects that you can only access using
     the proper accessor functions.

     NOTE! Opaqueness and "accessor functions" are not good in themselves.
     The reason we have them for things like pte_t etc is that there
     really is absolutely _zero_ portably accessible information there.

 (b) Clear integer types, where the abstraction _helps_ avoid confusion
     whether it is "int" or "long".

u8/u16/u32 are perfectly fine typedefs.

NOTE! Again - there needs to be a _reason_ for this. If something is "unsigned long", then there's no reason to do

typedef long myflags_t;

but if there is a clear reason for why it under certain circumstances might be an "unsigned int" and under other configurations might be "unsigned long", then by all means go ahead and use a typedef.

(c) when you use sparse to literally create a _new_ type for type-checking.

Maybe there are other cases too, but the rule should basically be to NEVER EVER use a typedef unless you can clearly match one of those rules.

In general, a pointer, or a struct that has elements that can reasonably be directly accessed should _never_ be a typedef.

Linus

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by dev on Fri, 03 Feb 2006 17:22:34 GMT
View Forum Message <> Reply to Message

Linus,

Not a problem and fully agree with you.
Just had to better review patch before sending.

Do you have any other ideas/comments on this?
I will send additional IPC/filesystems virtualization patches a bit later.

Kirill

> On Fri, 3 Feb 2006, Kirill Korotaev wrote:
>> This patch introduces some abstract container/VPS kernel structure and tiny
>> amount of operations on it.
>
> Please don't use things like "vps_t".
>
> It's a _mistake_ to use typedef for structures and pointers. When you see
> a
>
> vps_t a;

>
> in the source, what does it mean?
>
> In contrast, if it says
>
> struct virtual_container *a;
>
> you can actually tell what "a" is.
>
> Lots of people think that typedefs "help readability". Not so. They are
> useful only for
>
> (a) totally opaque objects (where the typedef is actively used to _hide_
>   what the object is).
>
>   Example: "pte_t" etc opaque objects that you can only access using
>   the proper accessor functions.
>
>   NOTE! Opaqueness and "accessor functions" are not good in themselves.
>   The reason we have them for things like pte_t etc is that there
>   really is absolutely _zero_ portably accessible information there.
>
> (b) Clear integer types, where the abstraction _helps_ avoid confusion
>   whether it is "int" or "long".
>
>   u8/u16/u32 are perfectly fine typedefs.
>
>   NOTE! Again - there needs to be a _reason_ for this. If something is
>   "unsigned long", then there's no reason to do
>
> typedef long myflags_t;
>
>   but if there is a clear reason for why it under certain circumstances
>   might be an "unsigned int" and under other configurations might be
>   "unsigned long", then by all means go ahead and use a typedef.
>
> (c) when you use sparse to literally create a _new_ type for
>   type-checking.
>
> Maybe there are other cases too, but the rule should basically be to NEVER
> EVER use a typedef unless you can clearly match one of those rules.
>
> In general, a pointer, or a struct that has elements that can reasonably
> be directly accessed should _never_ be a typedef.
>
>   Linus
>

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Linus Torvalds on Fri, 03 Feb 2006 17:49:56 GMT
View Forum Message <> Reply to Message

On Fri, 3 Feb 2006, Kirill Korotaev wrote:
>
> Do you have any other ideas/comments on this?
> I will send additional IPC/filesystems virtualization patches a bit later.

I think that a patch like this - particularly just the 1/5 part - makes total sense, because regardless of any other details of virtualization, every single scheme is going to need this.

So I think at least 1/5 (and quite frankly, 2-3/5 look that way too) are things that we can (and probably should) merge quickly, so that people can then actually look at the differences in the places that they may actually disagree about.

One thing I don't particularly like is some of the naming. To me "vps" doesn't sound particularly generic or logical. I realize that it probably makes perfect sense to you (and I assume it just means "virtual private servers"), but especially if you see patches 1-3 to really be independent of any "actual" virtualization code that is totally generic, I'd actually prefer a less specialized name.

I realize that what things like this is used for is VPS web hosting etc, but that's really naming by current common usage, not by any real "logic".

In other words, I personally would have called them "container" or something similar, rather than "vps_info". See? From a logical implementation standpoint, the fact that it is right now most commonly used for VPS hosting is totally irrelevant to the code, no?

(And hey, maybe your "vps" means something different. In which case my argument makes even more sense ;)

Linus

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Dave Hansen on Fri, 03 Feb 2006 18:34:01 GMT
View Forum Message <> Reply to Message

On Fri, 2006-02-03 at 09:49 -0800, Linus Torvalds wrote:
> One thing I don't particularly like is some of the naming. To me "vps"
> doesn't sound particularly generic or logical. I realize that it probably
> makes perfect sense to you (and I assume it just means "virtual private
> servers"), but especially if you see patches 1-3 to really be independent

> of any "actual" virtualization code that is totally generic, I'd actually
> prefer a less specialized name.

I just did a global s/vps/container/ and it looks pretty reasonable, at least from my point of view.

Couple of minor naming nitpick questions, though. Is vps/container_info really what we want to call it? It seems to me to be the basis for a real "container", without the _info part.

"tsk->owner_container" That makes it sound like a pointer to the "task owner's container". How about "owning_container"? The "container owning this task". Or, maybe just "container"?

Any particular reason for the "u32 id" in the vps_info struct as opposed to one of the more generic types? Do we want to abstract this one in the same way we do pid_t?

The "host" in "host_container_info" doesn't mean much to me. Though, I guess it has some context in the UML space. Would "init_container_info" or "root_container_info" be more descriptive?

Lastly, is this a place for krefs? I don't see a real need for a destructor yet, but the idea is fresh in my mind.

How does the attached patch look?

-- Dave

```
 memhotplug-dave/include/linux/container.h  |   40 ++++++++++++++++++++++++++++
 memhotplug-dave/include/linux/init_task.h  |    2 +
 memhotplug-dave/include/linux/sched.h      |    4 ++
 memhotplug-dave/init/main.c                |    8 +++++
 memhotplug-dave/kernel/exit.c              |    2 +
 memhotplug-dave/kernel/fork.c              |    2 +
 6 files changed, 58 insertions(+)

diff -puN include/linux/container.h~container-base include/linux/container.h
--- memhotplug/include/linux/container.h~container-base 2006-02-03 10:21:36.000000000 -0800
+++ memhotplug-dave/include/linux/container.h 2006-02-03 10:21:36.000000000 -0800
@@ -0,0 +1,40 @@
+#ifndef __LINUX_VPS_INFO_H_
+#define __LINUX_VPS_INFO_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 root_container;
+
+static inline container_t get_container(container_t container)
+{
+ atomic_inc(&container->refcnt);
+ return container;
+}
+
+static inline void put_container(container_t container)
+{
+ atomic_dec(&container->refcnt);
+}
+
+#include <linux/sched.h>
+#include <asm/current.h>
+
+static inline container_t set_current_container(container_t container)
+{
+ container_t ret;
+
+ ret = current->owning_container;
+ current->owning_container = container;
+ return ret;
+}
+
+#endif
diff -puN include/linux/init_task.h~container-base include/linux/init_task.h
--- memhotplug/include/linux/init_task.h~container-base 2006-02-03 10:21:36.000000000 -0800
+++ memhotplug-dave/include/linux/init_task.h 2006-02-03 10:23:49.000000000 -0800
@@ -3,6 +3,7 @@

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

 #define INIT_FDTABLE \
 {       \
@@ -121,6 +122,7 @@ extern struct group_info init_groups;
  .journal_info = NULL,      \
  .cpu_timers = INIT_CPU_TIMERS(tsk.cpu_timers),  \
  .fs_excl = ATOMIC_INIT(0),    \
```

```
+ .owning_container = &root_container,    \
 }


diff -puN include/linux/sched.h~container-base include/linux/sched.h
--- memhotplug/include/linux/sched.h~container-base 2006-02-03 10:21:36.000000000 -0800
+++ memhotplug-dave/include/linux/sched.h 2006-02-03 10:21:36.000000000 -0800
@@ -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 */
@@ -844,6 +845,7 @@ struct task_struct {
  struct backing_dev_info *backing_dev_info;

  struct io_context *io_context;
+ struct container *owning_container;

  unsigned long ptrace_message;
  siginfo_t *last_siginfo; /* For ptrace use.  */
@@ -874,6 +876,8 @@ struct task_struct {
  struct rcu_head rcu;
 };

+#define current_container() (current->owning_container)
+
 static inline pid_t process_group(struct task_struct *tsk)
 {
  return tsk->signal->pgrp;
diff -puN init/main.c~container-base init/main.c
--- memhotplug/init/main.c~container-base 2006-02-03 10:21:36.000000000 -0800
+++ memhotplug-dave/init/main.c 2006-02-03 10:21:36.000000000 -0800
@@ -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>
@@ -434,6 +435,13 @@ void __init parse_early_param(void)
  done = 1;
 }

+struct container root_container = {
```

```
+ .id  = 0,
+ .init_task = &init_task,
+ .refcnt  = ATOMIC_INIT(1),
+};
+
+EXPORT_SYMBOL(root_container);
 /*
  * Activate the first processor.
  */
diff -puN kernel/exit.c~container-base kernel/exit.c
--- memhotplug/kernel/exit.c~container-base 2006-02-03 10:21:36.000000000 -0800
+++ memhotplug-dave/kernel/exit.c 2006-02-03 10:21:36.000000000 -0800
@@ -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->owning_container);
  put_task_struct(p);

  p = leader;
diff -puN kernel/fork.c~container-base kernel/fork.c
--- memhotplug/kernel/fork.c~container-base 2006-02-03 10:21:36.000000000 -0800
+++ memhotplug-dave/kernel/fork.c 2006-02-03 10:21:36.000000000 -0800
@@ -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);
+ get_container(p->owning_container);
  if (unlikely(p->ptrace & PT_PTRACED))
   __ptrace_link(p, current->parent);

_
```

Subject: Summary: PID virtualization , Containers, Migration
Posted by Hubertus Franke on Fri, 03 Feb 2006 18:36:35 GMT
View Forum Message <> Reply to Message

Seems lots of interest is happening in this direction now
with various ideas and patches having been submitted.

I think the terms should be clarified a bit, even in our own
discussion we are using them loosely. Let's keep focusing
on the PID issues of containers and migration.

- user have come to expect that pids range from [0..PIDMAX]
- we can only migrate a self contained process tree (with some glue at the top)
- when we migrate we expect that the pid is preserved to the user
- we expect that UNIX semantics of pid visibility (e.g. kill etc) do
    not cross the container boundary (with some glue at the top)

Now there are two views to achieve the preservation that go at the heart
of the implementation only (not at the concept at all).
Two fundamental concepts are thrown around here:
(a) PID virtualization
(b) PID spaces

Let's be clear what I mean by these concepts:

PID virtualization:
==========
The kernel continues to use its current internal pid management.
Each task has a unique internal pid, throughout its life on that
particular kernel image.
At the user/kernel boundary these pids are virtualized on a
per container base. Hence pids coming from user space are converted to
an internal pid and internal pids handed to the user are converted
to the virtual pid.
As a result, all places along the user/kernel boundary that deal with
pids need to be identified and the conversion function needs to be called.
Also the vpid needs to be allocated/freed on fork/exit.

How this virtualization is maintained is an implementation detail.
One can keep a full "pid->vpid" and "vpi->pid" hash with the container
or play optimizations such as the OpenVZ folks to keep extra virtual
numbers in the task->pids[type] structure and or making
vpid == internal pid for non-containerized processes.

Right now the OpenVZ patch supplies a nice and clean means for this.
In a sense our patch also followes this approach because we explicitely
have these conversion functions and we too have similar optimizations
by masking the container id into the internal pid.
Other then on the details of the pid virtualization, the patches seem

to actually be the same ..

What has been pointed out is that
- calling these conversion function is overhead, cumbersome.
- confusing, hence dangerous, since we are maintaining different logical
    pid domains here under the same type pid_t

PID spaces:
==========

Pidspaces drive the isolation based on containers all the way down into the
current pid management. This at first sounds scary, but it will remove
lots of problem with the above approach.

Instead of keeping a single internal pid range and
its associated managment (aka pidspace), we provide one pidspace for each
container. All lookup functions operate on the container's pidspace,
hence this approach should remove the conversion functions and the two
logical domains, which seem to be big problem with the first approach.

Alan Cox made a very important statement, namely there is no reason that
we require unique pids in the kernel, even gave some examples where it isn't
true today. What becomes however a problem is the fact that now rather then
storing the pid as a reference, we need to now store the <pid,container> pair,
or more elegantly the pointer to the task. This later part unfortunately
has the problem of expired references to task structs.

I think that Eric's patch on weak task references, if I understood it correctly,
might come in handy. Instead of storing the task pointer, one stores the
task reference.

What needs to be solved is the container spawner transition into a new container
just like in the pid virtualization case.

Do people agree on these two characterizations and if so lets agree on what is
the right approach. From Linus's and Alan's comments I read they favor exploring
two, because of the problems listed with the first approach.

Other issues..

Herbert, brought up the issue of container hierarchy. Again, I think different
things come to mind here.

If indeed the containers are fully isolated from each other then its seems more

a lifecycle management. E.g. migration of a container implies migration of all
its children containers.
 From a kernel perspective, containers form a flat hierarchy.
 From a user's perspective one might allow certain operations (e.g. running the
process list of another container) and here constraints can be enforced.

e.g. ps --show-container mycontainer will show a specific containers

---

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Rik van Riel on Fri, 03 Feb 2006 18:36:56 GMT
View Forum Message <> Reply to Message

On Fri, 3 Feb 2006, Linus Torvalds wrote:

> So I think at least 1/5 (and quite frankly, 2-3/5 look that way too) are
> things that we can (and probably should) merge quickly, so that people
> can then actually look at the differences in the places that they may
> actually disagree about.

Agreed. There are a lot of common bits between the various
virtualization solutions, if we can get those out of the way
it should be a lot easier to get the last bits done...

--
All Rights Reversed

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Jeff Garzik on Fri, 03 Feb 2006 18:55:13 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Fri, 2006-02-03 at 09:49 -0800, Linus Torvalds wrote:
>
>>One thing I don't particularly like is some of the naming. To me "vps"
>>doesn't sound particularly generic or logical. I realize that it probably
>>makes perfect sense to you (and I assume it just means "virtual private
>>servers"), but especially if you see patches 1-3 to really be independent
>>of any "actual" virtualization code that is totally generic, I'd actually
>>prefer a less specialized name.
>
>
> I just did a global s/vps/container/ and it looks pretty reasonable, at
> least from my point of view.

I would have chosen the much shorter "box" or "jar", but that's just me :)

> "tsk->owner_container" That makes it sound like a pointer to the "task
> owner's container". How about "owning_container"? The "container
> owning this task". Or, maybe just "container"?

slip 'parent' in there...

> Any particular reason for the "u32 id" in the vps_info struct as opposed
> to one of the more generic types? Do we want to abstract this one in
> the same way we do pid_t?
>
> The "host" in "host_container_info" doesn't mean much to me. Though, I
> guess it has some context in the UML space. Would "init_container_info"
> or "root_container_info" be more descriptive?

probably

 Jeff

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Hubertus Franke on Fri, 03 Feb 2006 19:18:03 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Fri, 2006-02-03 at 09:49 -0800, Linus Torvalds wrote:
>
>>One thing I don't particularly like is some of the naming. To me "vps"
>>doesn't sound particularly generic or logical. I realize that it probably
>>makes perfect sense to you (and I assume it just means "virtual private
>>servers"), but especially if you see patches 1-3 to really be independent
>>of any "actual" virtualization code that is totally generic, I'd actually
>>prefer a less specialized name.
>
>
> I just did a global s/vps/container/ and it looks pretty reasonable, at
> least from my point of view.
>
> Couple of minor naming nitpick questions, though. Is vps/container_info
> really what we want to call it? It seems to me to be the basis for a
> real "container", without the _info part.
>
> "tsk->owner_container" That makes it sound like a pointer to the "task
> owner's container". How about "owning_container"? The "container
> owning this task". Or, maybe just "container"?
>

> Any particular reason for the "u32 id" in the vps_info struct as opposed
> to one of the more generic types? Do we want to abstract this one in
> the same way we do pid_t?
>
> The "host" in "host_container_info" doesn't mean much to me. Though, I
> guess it has some context in the UML space. Would "init_container_info"
> or "root_container_info" be more descriptive?
>
> Lastly, is this a place for krefs? I don't see a real need for a
> destructor yet, but the idea is fresh in my mind.
>
> How does the attached patch look?
>

Looks good to me ...

Similar to parts of our patch set, so overlap is good that means on
large parts we agree.
Let's go with Dave's adaption, since it already addresses some of
Linus's concerns and I start moving the pid isolation (in contrast to
pid virtualization) over this.

-- Hubertus

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Hubertus Franke on Fri, 03 Feb 2006 19:56:02 GMT
View Forum Message <> Reply to Message

missed the typedef on container ..
I am still on 2.6.15.2

> Couple of minor naming nitpick questions, though. Is vps/container_info
> really what we want to call it? It seems to me to be the basis for a
> real "container", without the _info part.
>
> "tsk->owner_container" That makes it sound like a pointer to the "task
> owner's container". How about "owning_container"? The "container
> owning this task". Or, maybe just "container"?
>
> Any particular reason for the "u32 id" in the vps_info struct as opposed
> to one of the more generic types? Do we want to abstract this one in
> the same way we do pid_t?
>
> The "host" in "host_container_info" doesn't mean much to me. Though, I
> guess it has some context in the UML space. Would "init_container_info"
> or "root_container_info" be more descriptive?
>

> Lastly, is this a place for krefs? I don't see a real need for a
> destructor yet, but the idea is fresh in my mind.
>
> How does the attached patch look?

here is that adapted ....

Index: linux-2.6.15/include/linux/container.h
 ===
--- /dev/null 1970-01-01 00:00:00.000000000 +0000
+++ linux-2.6.15/include/linux/container.h 2006-02-03 14:52:12.000000000
-0500
@@ -0,0 +1,42 @@
+#ifndef __LINUX_CONTAINER_H_
+#define __LINUX_CONTAINER_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 root_container;
+
+static inline struct container*
+get_container(struct container *container)
+{
+ atomic_inc(&container->refcnt);
+ return container;
+}
+
+static inline void put_container(struct container *container)
+{
+ atomic_dec(&container->refcnt);
+}
+
+#include <linux/sched.h>
+#include <asm/current.h>
+
+static inline struct container*
+set_current_container(struct container *container)
+{
+ struct container *ret;

```
+
+ ret = current->container;
+ current->container = container;
+ return ret;
+}
+
+#endif
Index: linux-2.6.15/include/linux/init_task.h
 ===================================================================
--- linux-2.6.15.orig/include/linux/init_task.h 2006-02-03
14:50:39.000000000 -0500
+++ linux-2.6.15/include/linux/init_task.h 2006-02-03 14:52:12.000000000
-0500
@@ -3,6 +3,7 @@

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

 #define INIT_FDTABLE \
 {       \
@@ -121,6 +122,7 @@ extern struct group_info init_groups;
  .journal_info = NULL,      \
  .cpu_timers = INIT_CPU_TIMERS(tsk.cpu_timers),  \
  .fs_excl = ATOMIC_INIT(0),    \
+ .container = &root_container,    \
 }


Index: linux-2.6.15/include/linux/sched.h
 ===================================================================
--- linux-2.6.15.orig/include/linux/sched.h 2006-02-03
14:50:39.000000000 -0500
+++ linux-2.6.15/include/linux/sched.h 2006-02-03 14:52:12.000000000
-0500
@@ -682,6 +682,7 @@ 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 */
@@ -830,6 +831,7 @@ struct task_struct {
  struct backing_dev_info *backing_dev_info;

  struct io_context *io_context;
+ struct container *container;
```

```
  unsigned long ptrace_message;
  siginfo_t *last_siginfo; /* For ptrace use.  */
@@ -859,6 +861,8 @@ struct task_struct {
  atomic_t fs_excl; /* holding fs exclusive resources */
 };

+#define current_container() (current->container)
+
 static inline pid_t process_group(struct task_struct *tsk)
 {
  return tsk->signal->pgrp;
Index: linux-2.6.15/init/main.c
 ===================================================================
--- linux-2.6.15.orig/init/main.c 2006-02-03 14:50:39.000000000 -0500
+++ linux-2.6.15/init/main.c 2006-02-03 14:52:12.000000000 -0500
@@ -47,8 +47,8 @@
 #include <linux/rmap.h>
 #include <linux/mempolicy.h>
 #include <linux/key.h>
+#include <linux/container.h>
 #include <net/sock.h>
-
 #include <asm/io.h>
 #include <asm/bugs.h>
 #include <asm/setup.h>
@@ -438,6 +438,13 @@ void __init parse_early_param(void)
  done = 1;
 }

+struct container root_container = {
+ .id  = 0,
+ .init_task = &init_task,
+ .refcnt  = ATOMIC_INIT(1),
+};
+
+EXPORT_SYMBOL(root_container);
 /*
  * Activate the first processor.
  */
Index: linux-2.6.15/kernel/exit.c
 ===================================================================
--- linux-2.6.15.orig/kernel/exit.c 2006-02-03 14:50:39.000000000 -0500
+++ linux-2.6.15/kernel/exit.c 2006-02-03 14:52:12.000000000 -0500
@@ -29,6 +29,7 @@
 #include <linux/syscalls.h>
 #include <linux/signal.h>
 #include <linux/cn_proc.h>
```

```
+#include <linux/container.h>

 #include <asm/uaccess.h>
 #include <asm/unistd.h>
@@ -106,6 +107,7 @@ repeat:
  spin_unlock(&p->proc_lock);
  proc_pid_flush(proc_dentry);
  release_thread(p);
+ put_container(p->container);
  put_task_struct(p);

  p = leader;
Index: linux-2.6.15/kernel/fork.c
 ===================================================================
--- linux-2.6.15.orig/kernel/fork.c 2006-02-03 14:50:39.000000000 -0500
+++ linux-2.6.15/kernel/fork.c 2006-02-03 14:52:12.000000000 -0500
@@ -43,6 +43,7 @@
 #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>
@@ -1121,6 +1122,7 @@ static task_t *copy_process(unsigned lon
  p->ioprio = current->ioprio;

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

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Greg KH on Fri, 03 Feb 2006 20:19:45 GMT

View Forum Message <> Reply to Message

On Fri, Feb 03, 2006 at 10:34:01AM -0800, Dave Hansen wrote:
>
> Lastly, is this a place for krefs? I don't see a real need for a
> destructor yet, but the idea is fresh in my mind.

Well, what happens when you drop the last reference to this container?
Right now, your patch doesn't cause anything to happen, and if that's
acceptable, then fine, you don't need to use a struct kref.

But if not, then why have a reference count at all? :)

thanks,

greg k-h

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Hubertus Franke on Fri, 03 Feb 2006 20:34:45 GMT
View Forum Message <> Reply to Message

Greg KH wrote:
> On Fri, Feb 03, 2006 at 10:34:01AM -0800, Dave Hansen wrote:
>
>>Lastly, is this a place for krefs? I don't see a real need for a
>>destructor yet, but the idea is fresh in my mind.
>
>
> Well, what happens when you drop the last reference to this container?
> Right now, your patch doesn't cause anything to happen, and if that's
> acceptable, then fine, you don't need to use a struct kref.
>
> But if not, then why have a reference count at all? :)
>
> thanks,
>
> greg k-h
>

Greg ...

In our pid virtualization patches we removed the object automatically.
I presume that Kirill does the same .... although the code for that is not released.

I am working on converting our pid stuff so to follow the recommendation of Linux/Alan
to just do <container,pid> isolation through pidspaces.

I could release an allocation/ before hand. Based on Kirill's 1/5 patch adoption
(which took Linus's & Dave's naming comments into account)

How do we want to create the container?
In our patch we did it through a /proc/container filesystem.
Which created the container object and then on fork/exec switched over.

How about an additional sys_exec_container(exec_args + "container_name").
This does all the work like exec, but creates new container
with name "..." and attaches task to new container.
If name exists, an error -EEXIST will be raised !

-- Hubertus

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by dev on Sun, 05 Feb 2006 14:52:53 GMT
View Forum Message <> Reply to Message

>> Do you have any other ideas/comments on this?
>> I will send additional IPC/filesystems virtualization patches a bit later.
>
> I think that a patch like this - particularly just the 1/5 part - makes
> total sense, because regardless of any other details of virtualization,
> every single scheme is going to need this.
>
> So I think at least 1/5 (and quite frankly, 2-3/5 look that way too) are
> things that we can (and probably should) merge quickly, so that people can
> then actually look at the differences in the places that they may actually
> disagree about.
Can we merge also proc/sysfs/network/netfilters virtualization?

> In other words, I personally would have called them "container" or
> something similar, rather than "vps_info". See? From a logical
> implementation standpoint, the fact that it is right now most commonly
> used for VPS hosting is totally irrelevant to the code, no?
>
> (And hey, maybe your "vps" means something different. In which case my
> argument makes even more sense ;)
virtual private sandbox :)

Actually, we call them "virtual environments" (VE) in OpenVZ. It is more
than abstract and have a nice brief name. If this suits you - I will be
happy to commit patches as is :)

other variants:
virtual context (vc, vctx),
virtual containers (vc).

I personally don't like "container", since it is too long and I see no
good abreviations for this...

Kirill

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by dev on Sun, 05 Feb 2006 15:05:44 GMT
View Forum Message <> Reply to Message

> I just did a global s/vps/container/ and it looks pretty reasonable, at
> least from my point of view.
>
> Couple of minor naming nitpick questions, though. Is vps/container_info

> really what we want to call it? It seems to me to be the basis for a
> real "container", without the _info part.
Can be ommited.

> "tsk->owner_container" That makes it sound like a pointer to the "task
> owner's container". How about "owning_container"? The "container
> owning this task". Or, maybe just "container"?
This is why I don't like "container" name.

Please, also note, in OpenVZ we have 2 pointers on task_struct:
One is owner of a task (owner_env), 2nd is a current context (exec_env).
exec_env pointer is used to avoid adding of additional argument to all
the functions where current context is required.

Linus, does such approach makes sense to you or you prefer us to add
additional args to functions where container pointer is needed? This
looks undersiable for embedded guys and increases stack usage/code size
when virtualization is off, which doesn't look good for me.

> Any particular reason for the "u32 id" in the vps_info struct as opposed
> to one of the more generic types? Do we want to abstract this one in
> the same way we do pid_t?
VPS ID is passed to/from user space APIs and when you have a cluster
with different archs and VPSs it is better to have something in common
for managing this.

> The "host" in "host_container_info" doesn't mean much to me. Though, I
> guess it has some context in the UML space. Would "init_container_info"
> or "root_container_info" be more descriptive?
init_container?
(Again, this is why container doesn't sound for me :))

> Lastly, is this a place for krefs? I don't see a real need for a
> destructor yet, but the idea is fresh in my mind.
I don't see much need for krefs, do you?
In OpenVZ we have 2-level refcounting (mentioned recently by Linus as in
mm). Process counter is used to decide when container should
collapse/cleanuped and real refcounter is used to free the structures
which can be referenced from somewhere else.

Also there is some probability that use of krefs will result in sysfs :)))

> How does the attached patch look?
I will resend all 5 patches tomorrow. And maybe more.

Kirill

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by dev on Sun, 05 Feb 2006 15:10:04 GMT
View Forum Message <> Reply to Message

> On Fri, Feb 03, 2006 at 10:34:01AM -0800, Dave Hansen wrote:
>> Lastly, is this a place for krefs? I don't see a real need for a
>> destructor yet, but the idea is fresh in my mind.
>
> Well, what happens when you drop the last reference to this container?
> Right now, your patch doesn't cause anything to happen, and if that's
> acceptable, then fine, you don't need to use a struct kref.
>
> But if not, then why have a reference count at all? :)

Please note, this patch introduces only small parts of it.
It doesn't introduce the code which creates containers/destroys them etc.

As I mentioned in another email:
In OpenVZ we have 2-level refcounting (mentioned recently by Linus as in
mm). Process counter is used to decide when container should
collapse/cleanuped and real refcounter is used to free the structures
which can be referenced from somewhere else.

Kirill

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by dev on Sun, 05 Feb 2006 15:11:54 GMT
View Forum Message <> Reply to Message

> How do we want to create the container?
> In our patch we did it through a /proc/container filesystem.
> Which created the container object and then on fork/exec switched over.
this doesn't look good for a full virtualization solution, since proc
should be virtualized as well :)

> How about an additional sys_exec_container(exec_args + "container_name").
> This does all the work like exec, but creates new container
> with name "..." and attaches task to new container.
> If name exists, an error -EEXIST will be raised !
Why do you need exec?

Kirill

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Hubertus Franke on Sun, 05 Feb 2006 15:39:48 GMT

Kirill Korotaev wrote:
>> How do we want to create the container?
>> In our patch we did it through a /proc/container filesystem.
>> Which created the container object and then on fork/exec switched over.
>
> this doesn't look good for a full virtualization solution, since proc
> should be virtualized as well :)

Just lazy's man's development version of a faked sys_call to create the container
without having to go through all architectures ...
Nothing permanent..

>
>> How about an additional sys_exec_container(exec_args +
>> "container_name").
>> This does all the work like exec, but creates new container
>> with name "..." and attaches task to new container.
>> If name exists, an error -EEXIST will be raised !
>
> Why do you need exec?

(a) how do you create/destroy a container
(b) how do you attach yourself to it?

-- Hubertus

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Benjamin Herrenschmid on Mon, 06 Feb 2006 00:56:31 GMT

On Fri, 2006-02-03 at 19:58 +0300, Kirill Korotaev wrote:

> +static inline vps_t get_vps(vps_t vps)
> +{
> + atomic_inc(&vps->refcnt);
> + return vps;
> +}
> +
> +static inline void put_vps(vps_t vps)
> +{
> + atomic_dec(&vps->refcnt);
> +}

I'm not too sure about the refcounting here .. you never destroy the

object ? Also, why introduce your own refcounting mecanism instead of using existing ones ? You could probably use at least a kref to get a nice refcount + destructor instead of home made atomics based. Maybe some higher level structure if you think it makes sense (not too sure what this virtualization stuff is about so I can't comment on what data structure is appropriate here).

Cheers,
Ben.

Subject: Re: [RFC][PATCH 3/5] Virtualization/containers: UTSNAME
Posted by ebiederm on Mon, 06 Feb 2006 08:21:13 GMT
View Forum Message <> Reply to Message

I am disturbed by the introduction of #defines like current_vps() and vps_utsname.

Magic lower case #defines are usually a bad idea.

These defines hide the cost of the operations you are performing.
At that point you might as well name the thing system_utsname
so you don't have to change the code.

And of course you failed to change several references to system_utsname.

Eric

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by ebiederm on Mon, 06 Feb 2006 08:31:49 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@openvz.org> writes:

> This patch introduces some abstract container/VPS kernel structure and tiny
> amount of operations on it.
>
> Patches following this one will be used for virtualization of some resources
> based on this container infrastructure, including those VPIDs patches I sent
> before.
>
> What is important here is that:
> - each VPS has unique ID
> - each process in kernel can belong to one VPS only

Next time could you please a diffstat for your patches,
Thanks

Eric

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by ebiederm on Mon, 06 Feb 2006 08:39:03 GMT
View Forum Message <> Reply to Message

Linus Torvalds <torvalds@osdl.org> writes:

> On Fri, 3 Feb 2006, Kirill Korotaev wrote:
>>
>> Do you have any other ideas/comments on this?
>> I will send additional IPC/filesystems virtualization patches a bit later.
>
> I think that a patch like this - particularly just the 1/5 part - makes
> total sense, because regardless of any other details of virtualization,
> every single scheme is going to need this.

I strongly disagree with this approach. I think Al Viro got it
right when he created a separate namespace for filesystems.

First this presumes an all or nothing interface. But that is not
what people are doing. Different people want different subsets
of the functionality. For the migration work I am doing having
multiple meanings for the same uid isn't interesting.

Secondly by implementing this in one big chunk there is no
migration path when you want to isolate an additional part of the
kernel interface.

So I really think an approach that allows for incremental progress
that allows for different subsets of this functionality to
be used is a better approach. In clone we already have
a perfectly serviceable interface for that and I have
seen no one refute that. I'm not sure I have seen anyone
get it though.

My apologies for the late reply I didn't see this thread until
just a couple of minutes ago. linux-kernel can be hard to
follow when you aren't cc'd.

Patches hopefully sometime in the next 24hours. So hopefully

conversation can be carried forward in a productive manner.

Eric

Subject: Re: [RFC][PATCH 3/5] Virtualization/containers: UTSNAME
Posted by dev on Mon, 06 Feb 2006 08:51:32 GMT
View Forum Message <> Reply to Message

> I am disturbed by the introduction of #defines like current_vps() and
> vps_utsname.
>
> Magic lower case #defines are usually a bad idea.
It is not magic defines, this is done intentionally.
You can take a more detailed view into OpenVZ sources, but the idea is
to make kernel compilable without virtualization.
When virtualization is OFF all this macros are defined to trivial
variables/defines which make it an old good kernel.

For example current_vps() should be (&init_vps), i.e. host system
environment only.

vps_utsname will be defined as system_utsname and so on.

> These defines hide the cost of the operations you are performing.
> At that point you might as well name the thing system_utsname
> so you don't have to change the code.
You mean to have variable and define with the same names?
it is not always good. It works fine, when both are defined in the same
file, but poorly when it is scattered all around...

> And of course you failed to change several references to
> system_utsname.
which one? Maybe intentionally? ;-)
Kirill

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by dev on Mon, 06 Feb 2006 08:58:39 GMT
View Forum Message <> Reply to Message

>>I think that a patch like this - particularly just the 1/5 part - makes
>>total sense, because regardless of any other details of virtualization,
>>every single scheme is going to need this.
> I strongly disagree with this approach. I think Al Viro got it
> right when he created a separate namespace for filesystems.
These patch set introduces separate namespaces as those in filesystems.

What exactly you don't like in this approach? Can you be more specific?

> First this presumes an all or nothing interface. But that is not
> what people are doing. Different people want different subsets
> of the functionality. For the migration work I am doing having
> multiple meanings for the same uid isn't interesting.
What do you mean by that? That you don't care about virtualization of
UIDs? So your migration doesn't care at all whether 2 systems have same
uids? Do you keep /etc/passwd in sync when do migration?
Only full virtualization allows to migrate applications without bugs and
different effects.

> Secondly by implementing this in one big chunk there is no
> migration path when you want to isolate an additional part of the
> kernel interface.
>
> So I really think an approach that allows for incremental progress
> that allows for different subsets of this functionality to
> be used is a better approach. In clone we already have
> a perfectly serviceable interface for that and I have
> seen no one refute that. I'm not sure I have seen anyone
> get it though.
Just introduce config option for each virtualization functionality.
That's it.

> My apologies for the late reply I didn't see this thread until
> just a couple of minutes ago. linux-kernel can be hard to
> follow when you aren't cc'd.
>
>
> Patches hopefully sometime in the next 24hours. So hopefully
> conversation can be carried forward in a productive manner.
Ok. I will remake them either :)

Kirill

Subject: Re: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by dev on Mon, 06 Feb 2006 09:01:02 GMT
View Forum Message <> Reply to Message

>>+static inline vps_t get_vps(vps_t vps)
>>+{
>>+ atomic_inc(&vps->refcnt);
>>+ return vps;
>>+}
>>+
>>+static inline void put_vps(vps_t vps)

>>+{
>>+ atomic_dec(&vps->refcnt);
>>+}
>
>
> I'm not too sure about the refcounting here .. you never destroy the
> object ? Also, why introduce your own refcounting mecanism instead of
> using existing ones ? You could probably use at least a kref to get a
> nice refcount + destructor instead of home made atomics based. Maybe
> some higher level structure if you think it makes sense (not too sure
> what this virtualization stuff is about so I can't comment on what data
> structure is appropriate here).
I replied to this in another email. Briefly:
this patch set doesn't introduce VPS/container creation/destroy
interface yet. Only small parts.
krefs are not needed here.

Kirill

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by dev on Mon, 06 Feb 2006 09:06:40 GMT
View Forum Message <> Reply to Message

> Just lazy's man's development version of a faked sys_call to create the
> container
> without having to go through all architectures ...
> Nothing permanent..

>>> How about an additional sys_exec_container(exec_args +
>>> "container_name").
>>> This does all the work like exec, but creates new container
>>> with name "..." and attaches task to new container.
>>> If name exists, an error -EEXIST will be raised !
>>
>>
>> Why do you need exec?
>
>
> (a) how do you create/destroy a container
> (b) how do you attach yourself to it?
a)
  create via syscall.
  destroyed when the last process dies in container.
b)
  syscall which changes container. you need to close unneeded
resources, change context and fork(). That's it.
  The whole process is exactly the same as if you changes UID.

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by ebiederm on Mon, 06 Feb 2006 09:19:37 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

>>> I think that a patch like this - particularly just the 1/5 part - makes total
>>> sense, because regardless of any other details of virtualization, every
>>> single scheme is going to need this.
>> I strongly disagree with this approach. I think Al Viro got it
>> right when he created a separate namespace for filesystems.

> These patch set introduces separate namespaces as those in filesystems.
> What exactly you don't like in this approach? Can you be more specific?

That you placed the namespaces in a separate structure from task_struct.
That part seems completely unnecessary, that and the addition of a
global id in a completely new namespace that will be a pain to virtualize
when it's time comes.

>> First this presumes an all or nothing interface. But that is not
>> what people are doing. Different people want different subsets
>> of the functionality. For the migration work I am doing having
>> multiple meanings for the same uid isn't interesting.

> What do you mean by that? That you don't care about virtualization of UIDs? So
> your migration doesn't care at all whether 2 systems have same uids? Do you keep
> /etc/passwd in sync when do migration?

It is already in sync. When the only interesting target is inside of
a cluster it is trivial to handle. In addition frequently there is
only one job on a given machine at a time so there is no other
interesting user of UIDs.

> Only full virtualization allows to migrate applications without bugs and
> different effects.

The requirement is that for every global identifier you have a
namespace where you will not have a conflict in definition when you
restore that identifier. (Either that or you have to update all
references to that identifier which is usually to hideous to take
seriously)

Virtualization has nothing to do with it. It is all about
non-conflicting names. The kernel already has everything virtualized.

>> Secondly by implementing this in one big chunk there is no
>> migration path when you want to isolate an additional part of the
>> kernel interface.

>> So I really think an approach that allows for incremental progress
>> that allows for different subsets of this functionality to
>> be used is a better approach. In clone we already have
>> a perfectly serviceable interface for that and I have
>> seen no one refute that. I'm not sure I have seen anyone
>> get it though.
> Just introduce config option for each virtualization functionality. That's it.

If you have different pointers for each one that is sane. I meant in
the api. I did not see patches 4/5 and 5/5. But I presumed since
you allocated a great global structure everything was turned on
off with that.

>> My apologies for the late reply I didn't see this thread until
>> just a couple of minutes ago. linux-kernel can be hard to
>> follow when you aren't cc'd.
>> Patches hopefully sometime in the next 24hours. So hopefully
>> conversation can be carried forward in a productive manner.
> Ok. I will remake them either :)

My not seeing your user space api is one of the problems with incomplete
patches which seem to plague this conversation.

Eric

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Dave Hansen on Mon, 06 Feb 2006 16:35:33 GMT
View Forum Message <> Reply to Message

On Sun, 2006-02-05 at 18:05 +0300, Kirill Korotaev wrote:
> > "tsk->owner_container" That makes it sound like a pointer to the "task
> > owner's container". How about "owning_container"? The "container
> > owning this task". Or, maybe just "container"?
> This is why I don't like "container" name.

I worry that using something like "vps" obfuscates the real meaning a
bit. The reason that "owner_vps" doesn't sound weird is that people, by
default, usually won't understand what a "vps" is.

(if you like acronyms a lot, I'm sure I can find a job for you at IBM or
in the US military :)

> Please, also note, in OpenVZ we have 2 pointers on task_struct:
> One is owner of a task (owner_env), 2nd is a current context (exec_env).
> exec_env pointer is used to avoid adding of additional argument to all
> the functions where current context is required.

That makes sense. However, are there many cases in the kernel where a task ends up doing something temporary like this:

```
tsk->exec_vnc = bar;
do_something_here(task);
tsk->exec_vnc = foo;
```

If that's the case very often, we probably want to change the APIs, just to make the common action explicit. If it never happens, or is a rarity, I think it should be just fine.

> > Any particular reason for the "u32 id" in the vps_info struct as opposed
> > to one of the more generic types? Do we want to abstract this one in
> > the same way we do pid_t?
> VPS ID is passed to/from user space APIs and when you have a cluster
> with different archs and VPSs it is better to have something in common
> for managing this.

I guess it does keep you from running into issues with mixing 32 and 64-bit processes. But, haven't we solved those problems already? Is it just a pain?

> > Lastly, is this a place for krefs? I don't see a real need for a
> > destructor yet, but the idea is fresh in my mind.
> I don't see much need for krefs, do you?
> In OpenVZ we have 2-level refcounting (mentioned recently by Linus as in
> mm). Process counter is used to decide when container should
> collapse/cleanuped and real refcounter is used to free the structures
> which can be referenced from somewhere else.

It sounds to me like anything that needs to have an action taken when a refcount reaches zero is a good candidate for a kref. Both of those uses sound like they need that. Probably not too big of a deal, though.

-- Dave

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Dave Hansen on Mon, 06 Feb 2006 16:37:54 GMT
View Forum Message <> Reply to Message

On Mon, 2006-02-06 at 02:19 -0700, Eric W. Biederman wrote:
> That you placed the namespaces in a separate structure from
> task_struct.
> That part seems completely unnecessary, that and the addition of a
> global id in a completely new namespace that will be a pain to
> virtualize
> when it's time comes.

Could you explain a bit why the container ID would need to be
virtualized?

-- Dave

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by dev on Mon, 06 Feb 2006 16:50:08 GMT
View Forum Message <> Reply to Message

Hello,

> I worry that using something like "vps" obfuscates the real meaning a
> bit. The reason that "owner_vps" doesn't sound weird is that people, by
> default, usually won't understand what a "vps" is.
container or context sounds the same :) it is impossible to feel this
notion naturally without getting into details. IMHO.

> (if you like acronyms a lot, I'm sure I can find a job for you at IBM or
> in the US military :)
We can talk about it separetely :)))

>>Please, also note, in OpenVZ we have 2 pointers on task_struct:
>>One is owner of a task (owner_env), 2nd is a current context (exec_env).
>>exec_env pointer is used to avoid adding of additional argument to all
>>the functions where current context is required.
>
> That makes sense. However, are there many cases in the kernel where a
> task ends up doing something temporary like this:
>
> tsk->exec_vnc = bar;
> do_something_here(task);
> tsk->exec_vnc = foo;
>
> If that's the case very often, we probably want to change the APIs, just
> to make the common action explicit. If it never happens, or is a
> rarity, I think it should be just fine.
It is quite rare. In IRQ, softIRQ, TCP/IP stack and some timers. Not much.

>>VPS ID is passed to/from user space APIs and when you have a cluster
>>with different archs and VPSs it is better to have something in common
>>for managing this.
> I guess it does keep you from running into issues with mixing 32 and
> 64-bit processes. But, haven't we solved those problems already? Is it
> just a pain?
VPSs can live in clusters. It is good to have one VPS ID space.

Kirill

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Linus Torvalds on Mon, 06 Feb 2006 16:56:52 GMT
View Forum Message <> Reply to Message

On Sun, 5 Feb 2006, Kirill Korotaev wrote:
>
> Please, also note, in OpenVZ we have 2 pointers on task_struct:
> One is owner of a task (owner_env), 2nd is a current context (exec_env).
> exec_env pointer is used to avoid adding of additional argument to all the
> functions where current context is required.

That naming _has_ to change.

"exec" has a very clear meaning in unix: it talks about the notion of
switching to another process image, or perhaps the bit that says that a
file contains an image that can be executed. It has nothing to do with
"current".

What you seem to be talking about is the _effective_ environment. Ie the
same way we have "uid" and "euid", you'd have a "container" and the
"effective container".

The "owner" name also makes no sense. The security context doesn't "own"
tasks. A task is _part_ of a context.

So if some people don't like "container", how about just calling it
"context"? The downside of that name is that it's very commonly used in
the kenel, because a lot of things have "contexts". That's why "container"
would be a lot better.

I'd suggest

 current->container - the current EFFECTIVE container
 current->master_container - the "long term" container.

(replace "master" with some other non-S&M term if you want)

(It would make sense to just have the prepend-"e" semantics of uid/gid,
but the fact is, "euid/egid" has a long unix history and is readable only
for that reason. The same wouldn't be true of containers. And
"effective_container" is probably too long to use for the field that is
actually the _common_ case. Thus the above suggestion).

  Linus

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by dev on Mon, 06 Feb 2006 17:19:54 GMT
View Forum Message <> Reply to Message

>>Please, also note, in OpenVZ we have 2 pointers on task_struct:
>>One is owner of a task (owner_env), 2nd is a current context (exec_env).
>>exec_env pointer is used to avoid adding of additional argument to all the
>>functions where current context is required.
>
>
> That naming _has_ to change.
I agree.

> "exec" has a very clear meaning in unix: it talks about the notion of
> switching to another process image, or perhaps the bit that says that a
> file contains an image that can be executed. It has nothing to do with
> "current".
> What you seem to be talking about is the _effective_ environment. Ie the
> same way we have "uid" and "euid", you'd have a "container" and the
> "effective container".
agree on this either. Good point.

> The "owner" name also makes no sense. The security context doesn't "own"
> tasks. A task is _part_ of a context.

> So if some people don't like "container", how about just calling it
> "context"? The downside of that name is that it's very commonly used in
> the kenel, because a lot of things have "contexts". That's why "container"
> would be a lot better.
>
> I'd suggest
>
> current->container - the current EFFECTIVE container
> current->master_container - the "long term" container.
>
> (replace "master" with some other non-S&M term if you want)
maybe task_container? i.e. where task actually is.
Sounds good for you?

The only problem with such names I see, that task will be an exception
then compared to other objects. I mean, on other objects field
"container" will mean the container which object is part of. But for
tasks this will mean effective one. Only tasks need these 2 containers
pointers and I would prefer having the common one to be called simply
"container".

Maybe then
current->econtainer - effective container
current->container - "long term" container

> (It would make sense to just have the prepend-"e" semantics of uid/gid,
> but the fact is, "euid/egid" has a long unix history and is readable only
> for that reason. The same wouldn't be true of containers. And
> "effective_container" is probably too long to use for the field that is
> actually the _common_ case. Thus the above suggestion).
Your proposal looks quite nice.
Then we will have eventually "container" field on objects (not on task
only) which sounds good to me. I will prepare patches right now.

Kirill

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by ebiederm on Mon, 06 Feb 2006 18:37:07 GMT
View Forum Message <> Reply to Message

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

> On Mon, 2006-02-06 at 02:19 -0700, Eric W. Biederman wrote:
>> That you placed the namespaces in a separate structure from
>> task_struct.
>> That part seems completely unnecessary, that and the addition of a
>> global id in a completely new namespace that will be a pain to
>> virtualize
>> when it's time comes.
>
> Could you explain a bit why the container ID would need to be
> virtualized?

As someone said to me a little bit ago, for migration or checkpointing
ultimately you have to capture the entire user/kernel interface if
things are going to work properly. Now if we add this facility to
the kernel and it is a general purpose facility. It is only a matter
of time before we need to deal with nested containers.

Not considering the case of having nested containers now is just foolish.
Maybe we don't have to implement it yet but not considering it is silly.

As far as I can tell there is a very reasonable chance that when we
are complete there is a very reasonable chance that software suspend
will just be a special case of migration, done complete in user space.
That is one of the more practical examples I can think of where this
kind of functionality would be used.

Eric

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by dev on Mon, 06 Feb 2006 19:30:31 GMT
View Forum Message <> Reply to Message

> As someone said to me a little bit ago, for migration or checkpointing
> ultimately you have to capture the entire user/kernel interface if
> things are going to work properly. Now if we add this facility to
> the kernel and it is a general purpose facility. It is only a matter
> of time before we need to deal with nested containers.
Fully virtualized container is not a matter of virtualized ID - it is
the easiest thing to do actually, but a whole global problem of other
resources virtualization. We can ommit ID for now, if you like it more.

> Not considering the case of having nested containers now is just foolish.
> Maybe we don't have to implement it yet but not considering it is silly.
No one told that it is not considered. In fact PID virtualization send
both by IBM/us is abstract and doesn't care whether containers are
nested or not.

> As far as I can tell there is a very reasonable chance that when we
> are complete there is a very reasonable chance that software suspend
> will just be a special case of migration, done complete in user space.
> That is one of the more practical examples I can think of where this
> kind of functionality would be used.

Kirill

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Cedric Le Goater on Mon, 06 Feb 2006 22:31:49 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
>> How do we want to create the container?
>> In our patch we did it through a /proc/container filesystem.
>> Which created the container object and then on fork/exec switched over.
>
> this doesn't look good for a full virtualization solution, since proc
> should be virtualized as well :)

Well, /proc should be "virtualized" or "isolated", how do you expect a
container to work correctly ? plenty of user space tools depend on it.

C.

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup

Posted by Cedric Le Goater on Mon, 06 Feb 2006 22:40:55 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:

> As someone said to me a little bit ago, for migration or checkpointing
> ultimately you have to capture the entire user/kernel interface if
> things are going to work properly. Now if we add this facility to
> the kernel and it is a general purpose facility. It is only a matter
> of time before we need to deal with nested containers.
>
> Not considering the case of having nested containers now is just foolish.
> Maybe we don't have to implement it yet but not considering it is silly.

That could be restricted. Today, process groups are not nested. Why do you think nested containers are inevitable ?

> As far as I can tell there is a very reasonable chance that when we
> are complete there is a very reasonable chance that software suspend
> will just be a special case of migration, done complete in user space.

Being able to sofware suspend one container among many would be a very interesting feature to have.

C.

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Sam Vilain on Tue, 07 Feb 2006 00:28:08 GMT
View Forum Message <> Reply to Message

Linus Torvalds wrote:
> So if some people don't like "container", how about just calling it
> "context"? The downside of that name is that it's very commonly used in
> the kenel, because a lot of things have "contexts". That's why "container"
> would be a lot better.
>
> I'd suggest
>
> current->container - the current EFFECTIVE container
> current->master_container - the "long term" container.
>
> (replace "master" with some other non-S&M term if you want)

Hmm. You actually need a linked list, otherwise you have replaced a one level flat structure with a two level one, and you miss out on some of the applications. VServer uses a special structure for this.

Sam.

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by ebiederm on Tue, 07 Feb 2006 01:57:05 GMT
View Forum Message <> Reply to Message

Cedric Le Goater <clg@fr.ibm.com> writes:

> Eric W. Biederman wrote:
>
>> As someone said to me a little bit ago, for migration or checkpointing
>> ultimately you have to capture the entire user/kernel interface if
>> things are going to work properly. Now if we add this facility to
>> the kernel and it is a general purpose facility. It is only a matter
>> of time before we need to deal with nested containers.
>>
>> Not considering the case of having nested containers now is just foolish.
>> Maybe we don't have to implement it yet but not considering it is silly.
>
> That could be restricted. Today, process groups are not nested. Why do you
> think nested containers are inevitable ?

process groups are a completely different kind of beast.
A closer analogy are hierarchical name spaces and mounts.
If we didn't need things like waitpid outside one pid namespace
to wait for a nested namespace they would be complete disjoint
and the implementation would be trivial.

>> As far as I can tell there is a very reasonable chance that when we
>> are complete there is a very reasonable chance that software suspend
>> will just be a special case of migration, done complete in user space.
>
> Being able to sofware suspend one container among many would be a very
> interesting feature to have.

That is what checkpointing. And that is simply the persistent form of
migration.

Eric

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by dev on Tue, 07 Feb 2006 12:19:14 GMT
View Forum Message <> Reply to Message

>> So if some people don't like "container", how about just calling it
>> "context"? The downside of that name is that it's very commonly used
>> in the kenel, because a lot of things have "contexts". That's why
>> "container" would be a lot better.
>>
>> I'd suggest
>>
>> current->container - the current EFFECTIVE container
>> current->master_container - the "long term" container.
>>
>> (replace "master" with some other non-S&M term if you want)
>
>
> Hmm. You actually need a linked list, otherwise you have replaced a one
> level flat structure with a two level one, and you miss out on some of
> the applications. VServer uses a special structure for this.
Nope! :) This is pointer to current/effective container, which can be
anywhere in the hierarchy. list should be inside container struct.

Kirill

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by dev on Tue, 07 Feb 2006 12:25:51 GMT
View Forum Message <> Reply to Message

>>>How do we want to create the container?
>>>In our patch we did it through a /proc/container filesystem.
>>>Which created the container object and then on fork/exec switched over.
>>
>>this doesn't look good for a full virtualization solution, since proc
>>should be virtualized as well :)
>
>
> Well, /proc should be "virtualized" or "isolated", how do you expect a
> container to work correctly ? plenty of user space tools depend on it.
Sorry, not actually understand your question... :(
There is not much problems with virtualization of proc. It can be
virtualized correctly, so that tools are still working. For example, in
OpenVZ /proc has 2 trees - global and local.
global tree contains the entries which are visiable in all containers.
and local tree - only those which are visible to containers.
PIDs are shown also only those which present in container.

Kirill

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Sam Vilain on Tue, 07 Feb 2006 22:21:57 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
>>> I'd suggest
>>>
>>> current->container - the current EFFECTIVE container
>>> current->master_container - the "long term" container.
>>>
>>> (replace "master" with some other non-S&M term if you want)
>> Hmm. You actually need a linked list, otherwise you have replaced a one
>> level flat structure with a two level one, and you miss out on some of
>> the applications. VServer uses a special structure for this.
>
> Nope! :) This is pointer to current/effective container, which can be
> anywhere in the hierarchy. list should be inside container struct.

So why store anything other than the effective container in the task?

Sam.

Subject: swsusp done by migration (was Re: [RFC][PATCH 1/5] Virtualization/containers: startup)
Posted by Pavel Machek on Wed, 08 Feb 2006 21:54:13 GMT
View Forum Message <> Reply to Message

Hi!

> > Could you explain a bit why the container ID would need to be
> > virtualized?
>
> As someone said to me a little bit ago, for migration or checkpointing
> ultimately you have to capture the entire user/kernel interface if
> things are going to work properly. Now if we add this facility to
> the kernel and it is a general purpose facility. It is only a matter
> of time before we need to deal with nested containers.
>
> Not considering the case of having nested containers now is just foolish.
> Maybe we don't have to implement it yet but not considering it is silly.
>
> As far as I can tell there is a very reasonable chance that when we
> are complete there is a very reasonable chance that software suspend
> will just be a special case of migration, done complete in user space.
> That is one of the more practical examples I can think of where this
> kind of functionality would be used.

Well, for now software suspend is done at very different level
(it snapshots complete kernel state), but being able to use
migration for this is certainly nice option.

BTW you could do whole-machine-migration now with uswsusp; but you'd
need identical hardware and it would take a bit long...

Pavel
--
Thanks, Sharp!

Subject: Re: swsusp done by migration (was Re: [RFC][PATCH 1/5] Virtualization/containers: startup)
Posted by ebiederm on Thu, 09 Feb 2006 18:20:13 GMT
View Forum Message <> Reply to Message

Pavel Machek <pavel@ucw.cz> writes:

> Well, for now software suspend is done at very different level
> (it snapshots complete kernel state), but being able to use
> migration for this is certainly nice option.
>
> BTW you could do whole-machine-migration now with uswsusp; but you'd
> need identical hardware and it would take a bit long...

Right part of the goal is with doing it as we are doing it is that we can
define what the interesting state is.

Replacing software suspend is not an immediate goal but I think it is
a worthy thing to target. In part because if we really can rip things
out of the kernel store them in a portable format and restore them
we will also have the ability to upgrade the kernel with out stopping
user space applications...

But being able to avoid the uninteresting parts, and having the policy
complete controlled outside the kernel are the big wins we are shooting for.

Eric

Subject: Re: swsusp done by migration (was Re: [RFC][PATCH 1/5] Virtualization/containers: startup)
Posted by Kyle Moffett on Fri, 10 Feb 2006 00:21:50 GMT
View Forum Message <> Reply to Message

On Feb 09, 2006, at 13:20, Eric W. Biederman wrote:

> Pavel Machek <pavel@ucw.cz> writes:
>> Well, for now software suspend is done at very different level (it
>> snapshots complete kernel state), but being able to use migration
>> for this is certainly nice option.
>>
>> BTW you could do whole-machine-migration now with uswsusp; but
>> you'd need identical hardware and it would take a bit long...
>
> Right part of the goal is with doing it as we are doing it is that
> we can define what the interesting state is.
>
> Replacing software suspend is not an immediate goal but I think it
> is a worthy thing to target. In part because if we really can rip
> things out of the kernel store them in a portable format and
> restore them we will also have the ability to upgrade the kernel
> with out stopping user space applications...
>
> But being able to avoid the uninteresting parts, and having the
> policy complete controlled outside the kernel are the big wins we
> are shooting for.

<wishful thinking>
I can see another extension to this functionality. With appropriate
changes it might also be possible to have a container exist across
multiple computers using some cluster code for synchronization and
fencing. The outermost container would be the system boot container,
and multiple inner containers would use some sort of network-
container-aware cluster filesystem to spread multiple vservers across
multiple servers, distributing CPU and network load appropriately.
</wishful thinking>

Cheers,
Kyle Moffett

--
I have yet to see any problem, however complicated, which, when you
looked at it in the right way, did not become still more complicated.
   -- Poul Anderson

Subject: Re: swsusp done by migration (was Re: [RFC][PATCH 1/5] Virtualization/containers: startup)
Posted by Sam Vilain on Fri, 10 Feb 2006 04:31:08 GMT
View Forum Message <> Reply to Message

Kyle Moffett wrote:
> <wishful thinking>
> I can see another extension to this functionality. With appropriate

> changes it might also be possible to have a container exist across
> multiple computers using some cluster code for synchronization and
> fencing. The outermost container would be the system boot container,
> and multiple inner containers would use some sort of network-
> container-aware cluster filesystem to spread multiple vservers across
> multiple servers, distributing CPU and network load appropriately.
> </wishful thinking>

Yeah. If you fudged/virtualised /dev/random, the system clock, etc you
could even have Tandem-style transparent High Availability.
</more wishful thinking>

Actually there is relatively little difference between a NUMA system and
a cluster...

Sam.

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by Nigel Cunningham on Fri, 10 Feb 2006 05:40:56 GMT
View Forum Message <> Reply to Message

Hi.

On Tuesday 07 February 2006 04:37, Eric W. Biederman wrote:
> Dave Hansen <haveblue@us.ibm.com> writes:
> > On Mon, 2006-02-06 at 02:19 -0700, Eric W. Biederman wrote:
> >> That you placed the namespaces in a separate structure from
> >> task_struct.
> >> That part seems completely unnecessary, that and the addition of a
> >> global id in a completely new namespace that will be a pain to
> >> virtualize
> >> when it's time comes.
> >
> > Could you explain a bit why the container ID would need to be
> > virtualized?
>
> As someone said to me a little bit ago, for migration or checkpointing
> ultimately you have to capture the entire user/kernel interface if
> things are going to work properly. Now if we add this facility to
> the kernel and it is a general purpose facility. It is only a matter
> of time before we need to deal with nested containers.
>
> Not considering the case of having nested containers now is just foolish.
> Maybe we don't have to implement it yet but not considering it is silly.
>
> As far as I can tell there is a very reasonable chance that when we
> are complete there is a very reasonable chance that software suspend

> will just be a special case of migration, done complete in user space.
> That is one of the more practical examples I can think of where this
> kind of functionality would be used.

Am I missing something? I though migration referred only to userspace
processes. Software suspend on the other hand, deals with the whole system,
of which process data/context is only a part.

Regards,

Nigel

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by ebiederm on Fri, 10 Feb 2006 06:01:26 GMT
View Forum Message <> Reply to Message

Nigel Cunningham <ncunningham@cyclades.com> writes:

> Am I missing something? I though migration referred only to userspace
> processes. Software suspend on the other hand, deals with the whole system,
> of which process data/context is only a part.

The problem domain is user process and the kernel state they depend on.
Implementation wise we are looking at two totally different problems.

However the effects should be similar if the set of processes to
migrate are all of the processes in the system.

For most of the interesting cases migration does not need to be that
ambitious.

Eric

Subject: Re: Re: swsusp done by migration (was Re: [RFC][PATCH 1/5]
Virtualization/containers: startup)
Posted by vaverin on Fri, 10 Feb 2006 06:23:56 GMT
View Forum Message <> Reply to Message

Sam Vilain wrote:
> Kyle Moffett wrote:
>> <wishful thinking>
>> I can see another extension to this functionality. With appropriate
>> changes it might also be possible to have a container exist across
>> multiple computers using some cluster code for synchronization and
>> fencing. The outermost container would be the system boot container,

>> and multiple inner containers would use some sort of network-
>> container-aware cluster filesystem to spread multiple vservers across
>> multiple servers, distributing CPU and network load appropriately.
>> </wishful thinking>
>
> Yeah. If you fudged/virtualised /dev/random, the system clock, etc you
> could even have Tandem-style transparent High Availability.
> </more wishful thinking>

Could you please explain, why you want to virtualize /dev/random?

Tnank you,
 Vasily Averin

Virtuozzo Linux Kernel Team

Subject: Re: swsusp done by migration (was Re: [RFC][PATCH 1/5] Virtualization/containers: startup)
Posted by Kyle Moffett on Fri, 10 Feb 2006 08:29:57 GMT
View Forum Message <> Reply to Message

On Feb 09, 2006, at 23:31, Sam Vilain wrote:
> Kyle Moffett wrote:
>> <wishful thinking>
>> I can see another extension to this functionality. With
>> appropriate changes it might also be possible to have a container
>> exist across multiple computers using some cluster code for
>> synchronization and fencing. The outermost container would be
>> the system boot container, and multiple inner containers would
>> use some sort of network- container-aware cluster filesystem to
>> spread multiple vservers across multiple servers, distributing
>> CPU and network load appropriately.
>> </wishful thinking>
>
> Yeah. If you fudged/virtualised /dev/random, the system clock, etc
> you could even have Tandem-style transparent High Availability.
> </more wishful thinking>
>
> Actually there is relatively little difference between a NUMA
> system and a cluster...

Yeah, a cluster is just a multi-tiered multi-address-space RNUMA
(*Really* Non-Uniform Memory Architecture) :-D. With some kind of
RDMA infiniband card and the right kernel and userspace tools, that
kind of cluster could be practical.

I _suspect_ (never really considered the issue before) that a

properly virtualized container could even achieve extremely high
fault tolerance by allowing systems to "vote" on correct output. If
you synchronize /dev/random and network IO across the system
correctly such that each instance of each userspace process on each
system sees _exactly_ the same virtual inputs and virtual clock in
the exact same order, then you could binary-compare the output of 3
different servers. If one didn't agree, it could be discarded and
marked as failing.

Cheers,
Kyle Moffett

--
There are two ways of constructing a software design. One way is to
make it so simple that there are obviously no deficiencies. And the
other way is to make it so complicated that there are no obvious
deficiencies. The first method is far more difficult.
-- C.A.R. Hoare

Subject: Re: Re: swsusp done by migration (was Re: [RFC][PATCH 1/5] Virtualization/containers: startup)
Posted by Sam Vilain on Sat, 11 Feb 2006 02:38:18 GMT
View Forum Message <> Reply to Message

On Fri, 2006-02-10 at 09:23 +0300, Vasily Averin wrote:
> >> <wishful thinking>
> >> I can see another extension to this functionality. With appropriate
> >> changes it might also be possible to have a container exist across
> >> multiple computers using some cluster code for synchronization and
> >> fencing. The outermost container would be the system boot container,
> >> and multiple inner containers would use some sort of network-
> >> container-aware cluster filesystem to spread multiple vservers across
> >> multiple servers, distributing CPU and network load appropriately.
> >> </wishful thinking>
> > Yeah. If you fudged/virtualised /dev/random, the system clock, etc you
> > could even have Tandem-style transparent High Availability.
> > </more wishful thinking>
> Could you please explain, why you want to virtualize /dev/random?

When checkpointing it is important to preserve all state. If you are
doing transparent highly available computing, you need to make sure all
system calls get the same answers in the clones. So you would need to
virtualise the entropy pool.

There are likely to be dozens of other quite hard problems in the way
first. Like I said, wishful thinking :-).

Sam.

Subject: Re: Re: swsusp done by migration (was Re: [RFC][PATCH 1/5] Virtualization/containers: startup)
Posted by vaverin on Sat, 11 Feb 2006 17:29:02 GMT
View Forum Message <> Reply to Message

Sam Vilain wrote:
> On Fri, 2006-02-10 at 09:23 +0300, Vasily Averin wrote:
>>>Yeah. If you fudged/virtualised /dev/random, the system clock, etc you
>>>could even have Tandem-style transparent High Availability.
>>></more wishful thinking>
>>Could you please explain, why you want to virtualize /dev/random?
>
> When checkpointing it is important to preserve all state. If you are
> doing transparent highly available computing, you need to make sure all
> system calls get the same answers in the clones. So you would need to
> virtualise the entropy pool.

>From my point of view it is important to preserve only all the determinated state.

Ok, lets we've checkpointed and saved current entropy pool. But we have not any
guarantee that pool will be in the same state at the moment of first access to
it after wakeuping. Because a new entropy can change it unpredictable.

Am I right?

Thank you,
 Vasily Averin

Virtuozzo Linux kernel Team

Subject: Re: [RFC][PATCH 1/5] Virtualization/containers: startup
Posted by dev on Mon, 20 Feb 2006 11:54:02 GMT
View Forum Message <> Reply to Message

> So why store anything other than the effective container in the task?
Effective container is used for temporary context change, e.g. when
processing interrupts and need to handle skb. it is effective container
for this code. just like get_fs()/set_fs() works.
Original container pointer is used for external process identification,
e.g. whether to show task in /proc in context of another task.

Kirill