Subject: [RFC] Virtualization patches for IPC/UTS. 2nd step
Posted by dev on Fri, 24 Mar 2006 17:23:39 GMT
View Forum Message <> Reply to Message

I propose to consider these patches for utsname and sysv IPC
virtualization once more.

The main change from the previous version is an introduction of separate
namespaces for each subsytem as suggested by Eric Biederman. People who
work on containers don't care actually, so I splitted a container into
namespaces.

The naming conventions used in these patches are (as example for IPC):

CONFIG_IPC_NS - per namespace config option
struct ipc_namespace - structure describing namepsace
ipc_ns   - names of var pointers, in task_struct etc.
init_ipc_ns - default host namespace

interfaces:
get_ipc_ns - refcountig interface
put_ipc_ns - refcountig interface
create_ipc_ns - create _empty_ namespace
clone_ipc_ns - clone current namespace, if applicable
free_ipc_ns - destroy namespace when refs are 0

Please, note, these patches do not introduce CONFIG_XXX_NS option in any
of Kconfigs, as it is questionable whether to have them scattered all
around or place in some menu "Virtual namespaces". But patches compile
and work fine both w/o and with it.

Also, please, note, that these patches do not virtualize corresponding
sysctls or proc parts of uts/ipc. I suppose this must be postponed as we
have no any consensus on /proc.

Some other minor differences from Eric/Dave patches:
- these patches heavily use current namespace context instead of
  bypassing additional argument to all functions where required.
- these patches compile to the old good kernel when namespaces are off.

Both patches are also available in GIT repo at:
http://git.openvz.org/pub/linux-2.6-openvz-ms/

Thanks,
Kirill

Subject: [RFC][PATCH 1/2] Virtualization of UTS
Posted by dev on Fri, 24 Mar 2006 17:31:19 GMT
View Forum Message <> Reply to Message

This patch introduces utsname namespace in system, which allows to have
different utsnames on the host.
Introduces config option CONFIG_UTS_NS and uts_namespace structure for this.

http://git.openvz.org/?p=linux-2.6-openvz-ms;a=commitdiff;h=216bb5e42c7eef7f1ed361244a60b1496e8bdf63

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

Kirill

```
--- a/include/linux/init_task.h
+++ b/include/linux/init_task.h
@@ -3,6 +3,7 @@

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

 #define INIT_FDTABLE \
 {       \
@@ -72,6 +73,12 @@

 extern struct group_info init_groups;

+#ifdef CONFIG_UTS_NS
+#define INIT_UTS_NS .uts_ns = &init_uts_ns,
+#else
+#define INIT_UTS_NS
+#endif
+
 /*
  *  INIT_TASK is used to set up the first task table, touch at
  * your own risk!. Base=0, limit=0x1fffff (=2MB)
@@ -121,6 +128,7 @@ extern struct group_info init_groups;
  .journal_info = NULL,      \
  .cpu_timers = INIT_CPU_TIMERS(tsk.cpu_timers),  \
  .fs_excl = ATOMIC_INIT(0),    \
+ INIT_UTS_NS       \
 }


--- a/include/linux/sched.h
+++ b/include/linux/sched.h
```

```
@@ -688,6 +688,7 @@ static inline void prefetch_stack(struct

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

 struct task_struct {
  volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
@@ -802,6 +803,9 @@ struct task_struct {
  struct files_struct *files;
 /* namespace */
  struct namespace *namespace;
+#ifdef CONFIG_UTS_NS
+ struct uts_namespace *uts_ns;
+#endif
 /* signal handlers */
  struct signal_struct *signal;
  struct sighand_struct *sighand;
--- a/include/linux/utsname.h
+++ b/include/linux/utsname.h
@@ -30,7 +30,36 @@ struct new_utsname {
  char domainname[65];
 };

+#ifdef CONFIG_UTS_NS
+#include <asm/atomic.h>
+
+struct uts_namespace {
+ atomic_t cnt;
+ struct new_utsname name;
+};
+
+extern struct uts_namespace *create_uts_ns(void);
+extern struct uts_namespace *clone_uts_ns(void);
+extern void free_uts_ns(struct uts_namespace *ns);
+
+static inline void get_uts_ns(struct uts_namespace *ns)
+{
+ atomic_inc(&ns->cnt);
+}
+
+static inline void put_uts_ns(struct uts_namespace *ns)
+{
+ if (atomic_dec_and_test(&ns->cnt))
+  free_uts_ns(ns);
+}
+
+#define system_utsname (current->uts_ns->name)
```

```
+extern struct uts_namespace init_uts_ns;
+#else
+#define get_uts_ns(ns) do { } while (0)
+#define put_uts_ns(ns) do { } while (0)
 extern struct new_utsname system_utsname;
+#endif

 extern struct rw_semaphore uts_sem;
 #endif
--- a/init/version.c
+++ b/init/version.c
@@ -17,6 +17,51 @@

 int version_string(LINUX_VERSION_CODE);

+#ifdef CONFIG_UTS_NS
+struct uts_namespace init_uts_ns = {
+ .cnt = ATOMIC_INIT(1),
+ .name = {
+  .sysname = UTS_SYSNAME,
+  .nodename = UTS_NODENAME,
+  .release = UTS_RELEASE,
+  .version = UTS_VERSION,
+  .machine = UTS_MACHINE,
+  .domainname = UTS_DOMAINNAME,
+ },
+};
+
+struct uts_namespace *create_uts_ns(void)
+{
+ struct uts_namespace *ns;
+
+ ns = kmalloc(sizeof(struct uts_namespace), GFP_KERNEL);
+ if (ns == NULL)
+  return NULL;
+
+ memset(&ns->name, 0, sizeof(ns->name));
+ atomic_set(&ns->cnt, 1);
+ return ns;
+}
+
+struct uts_namespace *clone_uts_ns(void)
+{
+ struct uts_namespace *ns, *cur;
+
+ ns = kmalloc(sizeof(struct uts_namespace), GFP_KERNEL);
+ if (ns == NULL)
+  return NULL;
```

```
+
+ cur = current->uts_ns;
+ memcpy(&ns->name, &cur->name, sizeof(cur->name));
+ atomic_set(&ns->cnt, 1);
+ return ns;
+}
+
+void free_uts_ns(struct uts_namespace *ns)
+{
+ kfree(ns);
+}
+#else
 struct new_utsname system_utsname = {
  .sysname = UTS_SYSNAME,
  .nodename = UTS_NODENAME,
@@ -27,6 +72,7 @@ struct new_utsname system_utsname = {
 };

 EXPORT_SYMBOL(system_utsname);
+#endif

 const char linux_banner[] =
  "Linux version " UTS_RELEASE " (" LINUX_COMPILE_BY "@"
--- a/kernel/exit.c
+++ b/kernel/exit.c
@@ -107,6 +107,7 @@ repeat:
  spin_unlock(&p->proc_lock);
  proc_pid_flush(proc_dentry);
  release_thread(p);
+ put_uts_ns(p->uts_ns);
  put_task_struct(p);

  p = leader;
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -1192,6 +1192,7 @@ static task_t *copy_process(unsigned lon
  }
  attach_pid(p, PIDTYPE_TGID, p->tgid);
  attach_pid(p, PIDTYPE_PID, p->pid);
+ get_uts_ns(p->uts_ns);

  nr_threads++;
  total_forks++;
--- a/kernel/sysctl.c
+++ b/kernel/sysctl.c
@@ -229,12 +229,18 @@ static ctl_table root_table[] = {
  { .ctl_name = 0 }
 };
```

```
+#ifdef CONFIG_UTS_NS
+#define sysctl_system_utsname (init_uts_ns.name)
+#else
+#define sysctl_system_utsname (system_utsname)
+#endif
+
 static ctl_table kern_table[] = {
  {
   .ctl_name = KERN_OSTYPE,
   .procname = "ostype",
-  .data  = system_utsname.sysname,
-  .maxlen  = sizeof(system_utsname.sysname),
+  .data  = sysctl_system_utsname.sysname,
+  .maxlen  = sizeof(sysctl_system_utsname.sysname),
   .mode  = 0444,
   .proc_handler = &proc_doutsstring,
   .strategy = &sysctl_string,
@@ -242,8 +248,8 @@ static ctl_table kern_table[] = {
  {
   .ctl_name = KERN_OSRELEASE,
   .procname = "osrelease",
-  .data  = system_utsname.release,
-  .maxlen  = sizeof(system_utsname.release),
+  .data  = sysctl_system_utsname.release,
+  .maxlen  = sizeof(sysctl_system_utsname.release),
   .mode  = 0444,
   .proc_handler = &proc_doutsstring,
   .strategy = &sysctl_string,
@@ -251,8 +257,8 @@ static ctl_table kern_table[] = {
  {
   .ctl_name = KERN_VERSION,
   .procname = "version",
-  .data  = system_utsname.version,
-  .maxlen  = sizeof(system_utsname.version),
+  .data  = sysctl_system_utsname.version,
+  .maxlen  = sizeof(sysctl_system_utsname.version),
   .mode  = 0444,
   .proc_handler = &proc_doutsstring,
   .strategy = &sysctl_string,
@@ -260,8 +266,8 @@ static ctl_table kern_table[] = {
  {
   .ctl_name = KERN_NODENAME,
   .procname = "hostname",
-  .data  = system_utsname.nodename,
-  .maxlen  = sizeof(system_utsname.nodename),
+  .data  = sysctl_system_utsname.nodename,
+  .maxlen  = sizeof(sysctl_system_utsname.nodename),
```

```
   .mode  = 0644,
   .proc_handler = &proc_doutsstring,
   .strategy = &sysctl_string,
@@ -269,8 +275,8 @@ static ctl_table kern_table[] = {
  {
   .ctl_name = KERN_DOMAINNAME,
   .procname = "domainname",
-  .data  = system_utsname.domainname,
-  .maxlen  = sizeof(system_utsname.domainname),
+  .data  = sysctl_system_utsname.domainname,
+  .maxlen  = sizeof(sysctl_system_utsname.domainname),
   .mode  = 0644,
   .proc_handler = &proc_doutsstring,
   .strategy = &sysctl_string,
```

Subject: [RFC][PATCH 2/2] Virtualization of IPC
Posted by dev on Fri, 24 Mar 2006 17:35:58 GMT
View Forum Message <> Reply to Message

This patch introduces IPC namespaces, which allow to create isolated IPC
users or containers.
Introduces CONFIG_IPC_NS and ipc_namespace structure.
It also uses current->ipc_ns as a pointer to current namespace, which
reduces places where additional argument to functions should be added.

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

From: Pavel Emelianov <xemul@openvz.org>
Date: Wed, 22 Mar 2006 13:15:23 +0000 (-0500)
Subject: Move ipc objects into separate namespace.
X-Git-Url:
http://git.openvz.org/?p=linux-2.6-openvz-ms;a=commitdiff;h=84a441429e7af425d27a8cda5beb70de5521a0be

Move ipc objects into separate namespace.

```
--- a/include/linux/init_task.h
+++ b/include/linux/init_task.h
@@ -4,6 +4,7 @@
 #include <linux/file.h>
 #include <linux/rcupdate.h>
 #include <linux/utsname.h>
+#include <linux/ipc.h>

 #define INIT_FDTABLE \
```

```
 {      \
@@ -79,6 +80,12 @@ extern struct group_info init_groups;
 #define INIT_UTS_NS
 #endif

+#ifdef CONFIG_IPC_NS
+#define INIT_IPC_NS .ipc_ns = &init_ipc_ns,
+#else
+#define INIT_IPC_NS
+#endif
+
 /*
  *  INIT_TASK is used to set up the first task table, touch at
  * your own risk!. Base=0, limit=0x1fffff (=2MB)
@@ -129,6 +136,7 @@ extern struct group_info init_groups;
  .cpu_timers = INIT_CPU_TIMERS(tsk.cpu_timers),  \
  .fs_excl = ATOMIC_INIT(0),    \
  INIT_UTS_NS       \
+ INIT_IPC_NS       \
 }


--- a/include/linux/ipc.h
+++ b/include/linux/ipc.h
@@ -70,6 +70,50 @@ struct kern_ipc_perm

 #endif /* __KERNEL__ */

+#include <linux/config.h>
+
+#ifdef CONFIG_IPC_NS
+#include <asm/atomic.h>
+
+struct ipc_ids;
+struct ipc_namespace {
+ atomic_t cnt;
+
+ struct ipc_ids *sem_ids;
+ int sem_ctls[4];
+ int used_sems;
+
+ struct ipc_ids *msg_ids;
+ int msg_ctlmax;
+ int msg_ctlmnb;
+ int msg_ctlmni;
+
+ struct ipc_ids *shm_ids;
+ size_t shm_ctlmax;
```

```
+ size_t  shm_ctlall;
+ int  shm_ctlmni;
+ int shm_total;
+};
+
+extern struct ipc_namespace init_ipc_ns;
+struct ipc_namespace *create_ipc_ns(void);
+void free_ipc_ns(struct ipc_namespace *ns);
+
+static inline void get_ipc_ns(struct ipc_namespace *ns)
+{
+ atomic_inc(&ns->cnt);
+}
+
+static inline void put_ipc_ns(struct ipc_namespace *ns)
+{
+ if (atomic_dec_and_test(&ns->cnt))
+  free_ipc_ns(ns);
+}
+#else
+#define get_ipc_ns(ns) do { } while (0)
+#define put_ipc_ns(ns) do { } while (0)
+#endif
+
 #endif /* _LINUX_IPC_H */


--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -689,6 +689,7 @@ static inline void prefetch_stack(struct
 struct audit_context;  /* See audit.c */
 struct mempolicy;
 struct uts_namespace;
+struct ipc_namespace;

 struct task_struct {
  volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
@@ -806,6 +807,9 @@ struct task_struct {
 #ifdef CONFIG_UTS_NS
  struct uts_namespace *uts_ns;
 #endif
+#ifdef CONFIG_IPC_NS
+ struct ipc_namespace *ipc_ns;
+#endif
 /* signal handlers */
  struct signal_struct *signal;
  struct sighand_struct *sighand;
--- a/ipc/msg.c
```

```
+++ b/ipc/msg.c
@@ -32,11 +32,6 @@
 #include <asm/uaccess.h>
 #include "util.h"

-/* sysctl: */
-int msg_ctlmax = MSGMAX;
-int msg_ctlmnb = MSGMNB;
-int msg_ctlmni = MSGMNI;
-
 /* one msg_receiver structure for each sleeping receiver */
 struct msg_receiver {
  struct list_head r_list;
@@ -63,8 +58,6 @@ struct msg_sender {
 static atomic_t msg_bytes = ATOMIC_INIT(0);
 static atomic_t msg_hdrs = ATOMIC_INIT(0);

-static struct ipc_ids msg_ids;
-
 #define msg_lock(id) ((struct msg_queue*)ipc_lock(&msg_ids,id))
 #define msg_unlock(msq) ipc_unlock(&(msq)->q_perm)
 #define msg_rmid(id) ((struct msg_queue*)ipc_rmid(&msg_ids,id))
@@ -79,8 +72,46 @@ static int newque (key_t key, int msgflg
 static int sysvipc_msg_proc_show(struct seq_file *s, void *it);
 #endif

+static struct ipc_ids msg_ids;
+
+#ifndef CONFIG_IPC_NS
+int msg_ctlmax = MSGMAX;
+int msg_ctlmnb = MSGMNB;
+int msg_ctlmni = MSGMNI;
+#else
+int init_msg_ns(struct ipc_namespace *ns)
+{
+ struct ipc_ids *ids;
+
+ ids = kmalloc(sizeof(struct ipc_ids), GFP_KERNEL);
+ if (ids == NULL)
+  return -ENOMEM;
+
+ ns->msg_ctlmax = MSGMAX;
+ ns->msg_ctlmnb = MSGMNB;
+ ns->msg_ctlmni = MSGMNI;
+ ns->msg_ids = ids;
+ ids->ns = ns;
+ ipc_init_ids(ids, ns->msg_ctlmni);
+ return 0;
```

```
+}
+
+void fini_msg_ns(struct ipc_namespace *ns)
+{
+ kfree(ns->msg_ids);
+}
+
+#define msg_ctlmax (current->ipc_ns->msg_ctlmax)
+#define msg_ctlmnb (current->ipc_ns->msg_ctlmnb)
+#define msg_ctlmni (current->ipc_ns->msg_ctlmni)
+#endif
+
 void __init msg_init (void)
 {
+#ifdef CONFIG_IPC_NS
+ init_ipc_ns.msg_ids = &msg_ids;
+ msg_ids.ns = &init_ipc_ns;
+#endif
  ipc_init_ids(&msg_ids,msg_ctlmni);
  ipc_init_proc_interface("sysvipc/msg",
     "       key      msqid perms      cbytes       qnum lspid lrpid   uid   gid  cuid  cgid      stime      rtime
     ctime\n",
@@ -88,6 +119,10 @@ void __init msg_init (void)
     sysvipc_msg_proc_show);
 }

+#ifdef CONFIG_IPC_NS
+#define msg_ids  (*(current->ipc_ns->msg_ids))
+#endif
+
 static int newque (key_t key, int msgflg)
 {
  int id;
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -86,7 +86,6 @@
  ipc_checkid(&sem_ids,&sma->sem_perm,semid)
 #define sem_buildid(id, seq) \
  ipc_buildid(&sem_ids, id, seq)
-static struct ipc_ids sem_ids;

 static int newary (key_t, int, int);
 static void freeary (struct sem_array *sma, int id);
@@ -106,17 +105,52 @@ static int sysvipc_sem_proc_show(struct
  *
  */

+static struct ipc_ids sem_ids;
```

```
+#ifndef CONFIG_IPC_NS
 int sem_ctls[4] = {SEMMSL, SEMMNS, SEMOPM, SEMMNI};
 #define sc_semmsl (sem_ctls[0])
 #define sc_semmns (sem_ctls[1])
 #define sc_semopm (sem_ctls[2])
 #define sc_semmni (sem_ctls[3])
-
 static int used_sems;
+#else
+int init_sem_ns(struct ipc_namespace *ns)
+{
+ struct ipc_ids *ids;
+
+ ids = kmalloc(sizeof(struct ipc_ids), GFP_KERNEL);
+ if (ids == NULL)
+  return -ENOMEM;
+
+ ns->sem_ctls[0] = SEMMSL;
+ ns->sem_ctls[1] = SEMMNS;
+ ns->sem_ctls[2] = SEMOPM;
+ ns->sem_ctls[3] = SEMMNI;
+ ns->used_sems = 0;
+ ns->sem_ids = ids;
+ ids->ns = ns;
+ ipc_init_ids(ids, ns->sem_ctls[3]);
+ return 0;
+}
+
+void fini_sem_ns(struct ipc_namespace *ns)
+{
+ kfree(ns->sem_ids);
+}
+
+#define sc_semmsl (current->ipc_ns->sem_ctls[0])
+#define sc_semmns (current->ipc_ns->sem_ctls[1])
+#define sc_semopm (current->ipc_ns->sem_ctls[2])
+#define sc_semmni (current->ipc_ns->sem_ctls[3])
+#define used_sems (current->ipc_ns->used_sems)
+#endif

 void __init sem_init (void)
 {
- used_sems = 0;
+#ifdef CONFIG_IPC_NS
+ init_ipc_ns.sem_ids = &sem_ids;
+ sem_ids.ns = &init_ipc_ns;
+#endif
  ipc_init_ids(&sem_ids,sc_semmni);
```

```
  ipc_init_proc_interface("sysvipc/sem",
     "       key      semid perms      nsems   uid   gid  cuid  cgid      otime      ctime\n",
@@ -124,6 +158,10 @@ void __init sem_init (void)
     sysvipc_sem_proc_show);
 }

+#ifdef CONFIG_IPC_NS
+#define sem_ids  (*(current->ipc_ns->sem_ids))
+#endif
+
 /*
  * Lockless wakeup algorithm:
  * Without the check/retry algorithm a lockless wakeup is possible:
--- a/ipc/shm.c
+++ b/ipc/shm.c
@@ -38,8 +38,6 @@
 static struct file_operations shm_file_operations;
 static struct vm_operations_struct shm_vm_ops;

-static struct ipc_ids shm_ids;
-
 #define shm_lock(id) ((struct shmid_kernel*)ipc_lock(&shm_ids,id))
 #define shm_unlock(shp) ipc_unlock(&(shp)->shm_perm)
 #define shm_get(id) ((struct shmid_kernel*)ipc_get(&shm_ids,id))
@@ -53,14 +51,48 @@ static void shm_close (struct vm_area_st
 static int sysvipc_shm_proc_show(struct seq_file *s, void *it);
 #endif

+static struct ipc_ids shm_ids;
+#ifndef CONFIG_IPC_NS
 size_t shm_ctlmax = SHMMAX;
 size_t  shm_ctlall = SHMALL;
 int  shm_ctlmni = SHMMNI;
+static int shm_total; /* total number of shared memory pages */
+#else
+int init_shm_ns(struct ipc_namespace *ns)
+{
+ struct ipc_ids *ids;

-static int shm_tot; /* total number of shared memory pages */
+ ids = kmalloc(sizeof(struct ipc_ids), GFP_KERNEL);
+ if (ids == NULL)
+  return -ENOMEM;
+
+ ns->shm_ctlmax = SHMMAX;
+ ns->shm_ctlall = SHMALL;
+ ns->shm_ctlmni = SHMMNI;
+ ns->shm_total = 0;
```

```
+ ns->shm_ids = ids;
+ ids->ns = ns;
+ ipc_init_ids(ids, 1);
+ return 0;
+}
+
+void fini_shm_ns(struct ipc_namespace *ns)
+{
+ kfree(ns->shm_ids);
+}
+
+#define shm_ctlmax (current->ipc_ns->shm_ctlmax)
+#define shm_ctlall (current->ipc_ns->shm_ctlall)
+#define shm_ctlmni (current->ipc_ns->shm_ctlmni)
+#define shm_total (current->ipc_ns->shm_total)
+#endif

 void __init shm_init (void)
 {
+#ifdef CONFIG_IPC_NS
+ init_ipc_ns.shm_ids = &shm_ids;
+ shm_ids.ns = &init_ipc_ns;
+#endif
  ipc_init_ids(&shm_ids, 1);
  ipc_init_proc_interface("sysvipc/shm",
     "       key      shmid perms       size  cpid  lpid nattch   uid   gid  cuid  cgid      atime      dtime
ctime\n",
@@ -68,6 +100,10 @@ void __init shm_init (void)
     sysvipc_shm_proc_show);
 }

+#ifdef CONFIG_IPC_NS
+#define shm_ids  (*(current->ipc_ns->shm_ids))
+#endif
+
 static inline int shm_checkid(struct shmid_kernel *s, int id)
 {
  if (ipc_checkid(&shm_ids,&s->shm_perm,id))
@@ -114,7 +150,15 @@ static void shm_open (struct vm_area_str
  */
 static void shm_destroy (struct shmid_kernel *shp)
 {
- shm_tot -= (shp->shm_segsz + PAGE_SIZE - 1) >> PAGE_SHIFT;
+#ifdef CONFIG_IPC_NS
+ struct ipc_namespace *ns, *cur;
+
+ ns = (struct ipc_namespace *)shp->shm_file->private_data;
+ cur = current->ipc_ns;
```

```
+ current->ipc_ns = ns;
+#endif
+
+ shm_total -= (shp->shm_segsz + PAGE_SIZE - 1) >> PAGE_SHIFT;
  shm_rmid (shp->id);
  shm_unlock(shp);
  if (!is_file_hugepages(shp->shm_file))
@@ -125,6 +169,10 @@ static void shm_destroy (struct shmid_ke
  fput (shp->shm_file);
  security_shm_free(shp);
  ipc_rcu_putref(shp);
+
+#ifdef CONFIG_IPC_NS
+ current->ipc_ns = cur;
+#endif
 }

 /*
@@ -167,8 +215,23 @@ static int shm_mmap(struct file * file,
  return ret;
 }

+#ifdef CONFIG_IPC_NS
+static int shm_release(struct inode *ino, struct file *file)
+{
+ struct ipc_namespace *ns;
+
+ ns = (struct ipc_namespace *)file->private_data;
+ file->private_data = NULL;
+ put_ipc_ns(ns);
+ return 0;
+}
+#endif
+
 static struct file_operations shm_file_operations = {
- .mmap = shm_mmap,
+ .mmap  = shm_mmap,
+#ifdef CONFIG_IPC_NS
+ .release = shm_release,
+#endif
 #ifndef CONFIG_MMU
  .get_unmapped_area = shmem_get_unmapped_area,
 #endif
@@ -196,7 +259,7 @@ static int newseg (key_t key, int shmflg
  if (size < SHMMIN || size > shm_ctlmax)
   return -EINVAL;

- if (shm_tot + numpages >= shm_ctlall)
```

```
+ if (shm_total + numpages >= shm_ctlall)
   return -ENOSPC;

  shp = ipc_rcu_alloc(sizeof(*shp));
@@ -249,11 +312,16 @@ static int newseg (key_t key, int shmflg
  shp->shm_file = file;
  file->f_dentry->d_inode->i_ino = shp->id;

+#ifdef CONFIG_IPC_NS
+ get_ipc_ns(current->ipc_ns);
+ file->private_data = current->ipc_ns;
+#endif
+
  /* Hugetlb ops would have already been assigned. */
  if (!(shmflg & SHM_HUGETLB))
   file->f_op = &shm_file_operations;

- shm_tot += numpages;
+ shm_total += numpages;
  shm_unlock(shp);
  return shp->id;

@@ -470,7 +538,7 @@ asmlinkage long sys_shmctl (int shmid, i
   down(&shm_ids.sem);
   shm_info.used_ids = shm_ids.in_use;
   shm_get_stat (&shm_info.shm_rss, &shm_info.shm_swp);
-  shm_info.shm_tot = shm_tot;
+  shm_info.shm_tot = shm_total;
   shm_info.swap_attempts = 0;
   shm_info.swap_successes = 0;
   err = shm_ids.max_id;
--- a/ipc/util.c
+++ b/ipc/util.c
@@ -65,7 +65,7 @@ __initcall(ipc_init);
  * array itself.
  */

-void __init ipc_init_ids(struct ipc_ids* ids, int size)
+void __ipc_init ipc_init_ids(struct ipc_ids* ids, int size)
 {
  int i;
  sema_init(&ids->sem,1);
@@ -96,6 +96,48 @@ void __init ipc_init_ids(struct ipc_ids*
   ids->entries->p[i] = NULL;
 }

+#ifdef CONFIG_IPC_NS
+struct ipc_namespace init_ipc_ns = {
```

```
+ .cnt = ATOMIC_INIT(1),
+};
+
+struct ipc_namespace *create_ipc_ns(void)
+{
+ struct ipc_namespace *ns;
+
+ ns = kmalloc(sizeof(struct ipc_namespace), GFP_KERNEL);
+ if (ns == NULL)
+  goto out;
+
+ if (init_sem_ns(ns))
+  goto out_sem;
+ if (init_msg_ns(ns))
+  goto out_msg;
+ if (init_shm_ns(ns))
+  goto out_shm;
+
+ atomic_set(&ns->cnt, 1);
+ return ns;
+
+out_shm:
+ fini_msg_ns(ns);
+out_msg:
+ fini_sem_ns(ns);
+out_sem:
+ kfree(ns);
+out:
+ return NULL;
+}
+
+void free_ipc_ns(struct ipc_namespace *ns)
+{
+ fini_sem_ns(ns);
+ fini_msg_ns(ns);
+ fini_shm_ns(ns);
+ kfree(ns);
+}
+#endif
+
 #ifdef CONFIG_PROC_FS
 static struct file_operations sysvipc_proc_fops;
 /**
@@ -229,7 +271,9 @@ int ipc_addid(struct ipc_ids* ids, struc
  }
  return -1;
 found:
- ids->in_use++;
```

```
+ if (++ids->in_use == 1)
+  get_ipc_ns(ids->ns);
+
  if (id > ids->max_id)
   ids->max_id = id;

@@ -276,7 +320,8 @@ struct kern_ipc_perm* ipc_rmid(struct ip
  ids->entries->p[lid] = NULL;
  if(p==NULL)
   BUG();
- ids->in_use--;
+ if (--ids->in_use)
+  put_ipc_ns(ids->ns);

  if (lid == ids->max_id) {
   do {
--- a/ipc/util.h
+++ b/ipc/util.h
@@ -8,6 +8,8 @@
 #ifndef _IPC_UTIL_H
 #define _IPC_UTIL_H

+#include <linux/config.h>
+
 #define USHRT_MAX 0xffff
 #define SEQ_MULTIPLIER (IPCMNI)

@@ -20,6 +22,7 @@ struct ipc_id_ary {
  struct kern_ipc_perm *p[0];
 };

+struct ipc_namespace;
 struct ipc_ids {
  int in_use;
  int max_id;
@@ -28,10 +31,25 @@ struct ipc_ids {
  struct semaphore sem;
  struct ipc_id_ary nullentry;
  struct ipc_id_ary* entries;
+#ifdef CONFIG_IPC_NS
+ struct ipc_namespace *ns;
+#endif
 };

+#ifdef CONFIG_IPC_NS
+int init_sem_ns(struct ipc_namespace *ns);
+int init_msg_ns(struct ipc_namespace *ns);
+int init_shm_ns(struct ipc_namespace *ns);
```

```
+void fini_sem_ns(struct ipc_namespace *ns);
+void fini_msg_ns(struct ipc_namespace *ns);
+void fini_shm_ns(struct ipc_namespace *ns);
+#define __ipc_init
+#else
+#define __ipc_init __init
+#endif
+
 struct seq_file;
-void __init ipc_init_ids(struct ipc_ids* ids, int size);
+void __ipc_init ipc_init_ids(struct ipc_ids* ids, int size);
 #ifdef CONFIG_PROC_FS
 void __init ipc_init_proc_interface(const char *path, const char *header,
        struct ipc_ids *ids,
--- a/kernel/exit.c
+++ b/kernel/exit.c
@@ -108,6 +108,7 @@ repeat:
  proc_pid_flush(proc_dentry);
  release_thread(p);
  put_uts_ns(p->uts_ns);
+ put_ipc_ns(p->ipc_ns);
  put_task_struct(p);

  p = leader;
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -1193,6 +1193,7 @@ static task_t *copy_process(unsigned lon
  attach_pid(p, PIDTYPE_TGID, p->tgid);
  attach_pid(p, PIDTYPE_PID, p->pid);
  get_uts_ns(p->uts_ns);
+ get_ipc_ns(p->ipc_ns);

  nr_threads++;
  total_forks++;
--- a/kernel/sysctl.c
+++ b/kernel/sysctl.c
@@ -93,6 +93,7 @@ extern char modprobe_path[];
 extern int sg_big_buff;
 #endif
 #ifdef CONFIG_SYSVIPC
+#ifndef CONFIG_IPC_NS
 extern size_t shm_ctlmax;
 extern size_t shm_ctlall;
 extern int shm_ctlmni;
@@ -100,6 +101,16 @@ extern int msg_ctlmax;
 extern int msg_ctlmnb;
 extern int msg_ctlmni;
 extern int sem_ctls[];
```

+#else
+#include <linux/ipc.h>
+#define shm_ctlmax (init_ipc_ns.shm_ctlmax)
+#define shm_ctlall (init_ipc_ns.shm_ctlall)
+#define shm_ctlmni (init_ipc_ns.shm_ctlmni)
+#define msg_ctlmax (init_ipc_ns.msg_ctlmax)
+#define msg_ctlmnb (init_ipc_ns.msg_ctlmnb)
+#define msg_ctlmni (init_ipc_ns.msg_ctlmni)
+#define sem_ctls (init_ipc_ns.sem_ctls)
+#endif
 #endif

 #ifdef __sparc__

Subject: Re: [RFC][PATCH 1/2] Virtualization of UTS
Posted by ebiederm on Fri, 24 Mar 2006 19:09:01 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

> This patch introduces utsname namespace in system, which allows to have
> different utsnames on the host.
> Introduces config option CONFIG_UTS_NS and uts_namespace structure for this.

Ok. It looks like we need to resolve the sysctl issues before we merge
either patch, into the stable kernel.

We also need to discuss the system call interface, as without one
the functionality is unusable :)

>
http://git.openvz.org/?p=linux-2.6-openvz-ms;a=commitdiff;h=216bb5e42c7eef7f1ed361244a60b1496e8bdf63
>
> Signed-Off-By: Pavel Emelianov <xemul@openvz.org>
> Signed-Off-By: Kirill Korotaev <dev@openvz.org>
>
> Kirill
> --- a/include/linux/init_task.h
> +++ b/include/linux/init_task.h
> @@ -3,6 +3,7 @@
>
> #include <linux/file.h>
> #include <linux/rcupdate.h>
> +#include <linux/utsname.h>
>
> #define INIT_FDTABLE \

```
> {       \
> @@ -72,6 +73,12 @@
>
>  extern struct group_info init_groups;
>
> +#ifdef CONFIG_UTS_NS
> +#define INIT_UTS_NS .uts_ns = &init_uts_ns,
> +#else
> +#define INIT_UTS_NS
> +#endif
> +
>  /*
>   *  INIT_TASK is used to set up the first task table, touch at
>   * your own risk!. Base=0, limit=0x1fffff (=2MB)
> @@ -121,6 +128,7 @@ extern struct group_info init_groups;
>   .journal_info = NULL,      \
>   .cpu_timers = INIT_CPU_TIMERS(tsk.cpu_timers),  \
>   .fs_excl = ATOMIC_INIT(0),    \
> + INIT_UTS_NS       \
>  }
>
>
> --- a/include/linux/sched.h
> +++ b/include/linux/sched.h
> @@ -688,6 +688,7 @@ static inline void prefetch_stack(struct
>
>  struct audit_context;  /* See audit.c */
>  struct mempolicy;
> +struct uts_namespace;
>
>  struct task_struct {
>   volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
> @@ -802,6 +803,9 @@ struct task_struct {
>   struct files_struct *files;
>  /* namespace */
>   struct namespace *namespace;
> +#ifdef CONFIG_UTS_NS
> + struct uts_namespace *uts_ns;
> +#endif
>  /* signal handlers */
>   struct signal_struct *signal;
>   struct sighand_struct *sighand;
> --- a/include/linux/utsname.h
> +++ b/include/linux/utsname.h
> @@ -30,7 +30,36 @@ struct new_utsname {
>   char domainname[65];
>  };
>
```

```
> +#ifdef CONFIG_UTS_NS
> +#include <asm/atomic.h>
> +
> +struct uts_namespace {
> + atomic_t cnt;
> + struct new_utsname name;
> +};
> +
> +extern struct uts_namespace *create_uts_ns(void);
> +extern struct uts_namespace *clone_uts_ns(void);
> +extern void free_uts_ns(struct uts_namespace *ns);
> +
> +static inline void get_uts_ns(struct uts_namespace *ns)
> +{
> + atomic_inc(&ns->cnt);
> +}
> +
> +static inline void put_uts_ns(struct uts_namespace *ns)
> +{
> + if (atomic_dec_and_test(&ns->cnt))
> +  free_uts_ns(ns);
> +}
> +
> +#define system_utsname (current->uts_ns->name)
> +extern struct uts_namespace init_uts_ns;
> +#else
> +#define get_uts_ns(ns) do { } while (0)
> +#define put_uts_ns(ns) do { } while (0)
>  extern struct new_utsname system_utsname;
> +#endif
>
>  extern struct rw_semaphore uts_sem;
>  #endif
> --- a/init/version.c
> +++ b/init/version.c
> @@ -17,6 +17,51 @@
>
>  int version_string(LINUX_VERSION_CODE);
>
> +#ifdef CONFIG_UTS_NS
> +struct uts_namespace init_uts_ns = {
> + .cnt = ATOMIC_INIT(1),
> + .name = {
> +  .sysname = UTS_SYSNAME,
> +  .nodename = UTS_NODENAME,
> +  .release = UTS_RELEASE,
> +  .version = UTS_VERSION,
> +  .machine = UTS_MACHINE,
```

> + .domainname = UTS_DOMAINNAME,
> + },
> +};

> +struct uts_namespace *create_uts_ns(void)
> +{
> + struct uts_namespace *ns;
> +
> + ns = kmalloc(sizeof(struct uts_namespace), GFP_KERNEL);
> + if (ns == NULL)
> + return NULL;
> +
> + memset(&ns->name, 0, sizeof(ns->name));
> + atomic_set(&ns->cnt, 1);
> + return ns;
> +}

Setting name to 0. Seems very wrong.

> +struct uts_namespace *clone_uts_ns(void)
> +{
> + struct uts_namespace *ns, *cur;
> +
> + ns = kmalloc(sizeof(struct uts_namespace), GFP_KERNEL);
> + if (ns == NULL)
> + return NULL;
> +
> + cur = current->uts_ns;
> + memcpy(&ns->name, &cur->name, sizeof(cur->name));
> + atomic_set(&ns->cnt, 1);
> + return ns;
> +}

Having both create_uts_ns and clone_uts_ns is redundant.

There is a reasonable argument for copying resetting
nodename and domainname to UTS_NODENAME, and UTS_DOMAINNAME,
when we create a new instance. As we almost certainly do not
want to continue to use the old ones and we need some value
in there. Plus some user space tools special case the default
UTS_NODENAME for setting the name from DHCP.

We can also do this from user space so it is not a huge deal
either way. We certainly need to copy the other fields
and not initialize them from the defaults because there is
kernel initialization code that modifies the defaults.

> +void free_uts_ns(struct uts_namespace *ns)

> +{
> + kfree(ns);
> +}
> +#else
> struct new_utsname system_utsname = {
>   .sysname = UTS_SYSNAME,
>   .nodename = UTS_NODENAME,
> @@ -27,6 +72,7 @@ struct new_utsname system_utsname = {
> };
>
> EXPORT_SYMBOL(system_utsname);
> +#endif

I am a little concerned about the maintenance issues with
placing having two identical initializers from system_utsname.

> const char linux_banner[] =
>   "Linux version " UTS_RELEASE " (" LINUX_COMPILE_BY "@"
> --- a/kernel/exit.c
> +++ b/kernel/exit.c
> @@ -107,6 +107,7 @@ repeat:
>   spin_unlock(&p->proc_lock);
>   proc_pid_flush(proc_dentry);
>   release_thread(p);
> + put_uts_ns(p->uts_ns);
>   put_task_struct(p);
>
>   p = leader;
> --- a/kernel/fork.c
> +++ b/kernel/fork.c
> @@ -1192,6 +1192,7 @@ static task_t *copy_process(unsigned lon
>   }
>   attach_pid(p, PIDTYPE_TGID, p->tgid);
>   attach_pid(p, PIDTYPE_PID, p->pid);
> + get_uts_ns(p->uts_ns);
>
>   nr_threads++;
>   total_forks++;

We don't need the tasklist_lock around get_uts_ns.
Please move it earlier.

> --- a/kernel/sysctl.c
> +++ b/kernel/sysctl.c
> @@ -229,12 +229,18 @@ static ctl_table root_table[] = {
>   { .ctl_name = 0 }
> };

Since we can disable CONFIG_SYSCTL we probably want to leave
this part for another patch. Although incremental progress
is probably ok.

```
> +#ifdef CONFIG_UTS_NS
> +#define sysctl_system_utsname (init_uts_ns.name)
> +#else
> +#define sysctl_system_utsname (system_utsname)
> +#endif
> +
>  static ctl_table kern_table[] = {
>   {
>    .ctl_name = KERN_OSTYPE,
>    .procname = "ostype",
> -  .data  = system_utsname.sysname,
> -  .maxlen  = sizeof(system_utsname.sysname),
> +  .data  = sysctl_system_utsname.sysname,
> +  .maxlen  = sizeof(sysctl_system_utsname.sysname),
>    .mode  = 0444,
>    .proc_handler = &proc_doutsstring,
>    .strategy = &sysctl_string,
> @@ -242,8 +248,8 @@ static ctl_table kern_table[] = {
>   {
>    .ctl_name = KERN_OSRELEASE,
>    .procname = "osrelease",
> -  .data  = system_utsname.release,
> -  .maxlen  = sizeof(system_utsname.release),
> +  .data  = sysctl_system_utsname.release,
> +  .maxlen  = sizeof(sysctl_system_utsname.release),
>    .mode  = 0444,
>    .proc_handler = &proc_doutsstring,
>    .strategy = &sysctl_string,
> @@ -251,8 +257,8 @@ static ctl_table kern_table[] = {
>   {
>    .ctl_name = KERN_VERSION,
>    .procname = "version",
> -  .data  = system_utsname.version,
> -  .maxlen  = sizeof(system_utsname.version),
> +  .data  = sysctl_system_utsname.version,
> +  .maxlen  = sizeof(sysctl_system_utsname.version),
>    .mode  = 0444,
>    .proc_handler = &proc_doutsstring,
>    .strategy = &sysctl_string,
> @@ -260,8 +266,8 @@ static ctl_table kern_table[] = {
>   {
>    .ctl_name = KERN_NODENAME,
>    .procname = "hostname",
> -  .data  = system_utsname.nodename,
```

> - .maxlen = sizeof(system_utsname.nodename),
> + .data = sysctl_system_utsname.nodename,
> + .maxlen = sizeof(sysctl_system_utsname.nodename),
>   .mode = 0644,
>   .proc_handler = &proc_doutsstring,
>   .strategy = &sysctl_string,
> @@ -269,8 +275,8 @@ static ctl_table kern_table[] = {
>   {
>   .ctl_name = KERN_DOMAINNAME,
>   .procname = "domainname",
> - .data = system_utsname.domainname,
> - .maxlen = sizeof(system_utsname.domainname),
> + .data = sysctl_system_utsname.domainname,
> + .maxlen = sizeof(sysctl_system_utsname.domainname),
>   .mode = 0644,
>   .proc_handler = &proc_doutsstring,
>   .strategy = &sysctl_string,

Subject: Re: [RFC][PATCH 2/2] Virtualization of IPC
Posted by Dave Hansen on Fri, 24 Mar 2006 19:13:20 GMT
View Forum Message <> Reply to Message

On Fri, 2006-03-24 at 20:35 +0300, Kirill Korotaev wrote:
> This patch introduces IPC namespaces, which allow to create isolated IPC
> users or containers.
> Introduces CONFIG_IPC_NS and ipc_namespace structure.
> It also uses current->ipc_ns as a pointer to current namespace, which
> reduces places where additional argument to functions should be added.

In three words, I think this has "too many #ifdefs".

The non-containerized or namespaced case should probably just be one,
static namespace variable that gets wrapped up in some nice #ifdefed
hlper functions.

For instance, instead of this:

+#ifdef CONFIG_IPC_NS
+#define msg_ids		(*(current->ipc_ns->msg_ids))
+#endif

Have

#ifdef CONFIG_IPC_NS
static inline struct ipc_namespace *current_ipc_ns(void)
{
 return current->ipc_ns;

```
}
#else
static inline struct ipc_namespace *current_ipc_ns(void)
{
 return &static_ipc_ns;
}
#endif
```

And use current_ipc_ns()->msg_ids. I can't imagine that gcc can't
figure that out and turn it back into effectively the same thing.

I really dislike the idea of replacing nice variables with macros that
add indirection. They really might fool people. Putting a function
there is much nicer.

Why avoid to passing these things around as function arguments? Doesn't
that make it more explicit what is going on, and where the indirection
is occurring? Does it also make refcounting and lifetime issues easier
to manage?

BTW, Did you see my version of this?

-- Dave

Subject: Re: [RFC][PATCH 1/2] Virtualization of UTS
Posted by dev on Fri, 24 Mar 2006 19:35:20 GMT
View Forum Message <> Reply to Message

>> This patch introduces utsname namespace in system, which allows to have
>> different utsnames on the host.
>> Introduces config option CONFIG_UTS_NS and uts_namespace structure for this.
>
> Ok. It looks like we need to resolve the sysctl issues before we merge
> either patch, into the stable kernel.
I disagree with you. Right now we can have sysctl and proc for init
namespaces only.
And when sysctl and proc are virtualized somehow, we can fix all these.
I simply don't expect /proc and sysctl to be done quickly. As we have
very different approaches. And there is no any consensus. Why not to
commit working/agreed parts then?

> We also need to discuss the system call interface, as without one
> the functionality is unusable :)
I also don't see why it can be separated. There is an API in namespaces,
and how it is mapped into syscalls is another question. At least it
doesn't prevent us from commiting virtualization itself, agree?

Thanks,
Kirill

Subject: Re: [RFC][PATCH 1/2] Virtualization of UTS
Posted by ebiederm on Fri, 24 Mar 2006 19:50:36 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

>>> This patch introduces utsname namespace in system, which allows to have
>>> different utsnames on the host.
>>> Introduces config option CONFIG_UTS_NS and uts_namespace structure for this.
>> Ok. It looks like we need to resolve the sysctl issues before we merge
>> either patch, into the stable kernel.
> I disagree with you. Right now we can have sysctl and proc for init namespaces
> only.
> And when sysctl and proc are virtualized somehow, we can fix all these.
> I simply don't expect /proc and sysctl to be done quickly. As we have very
> different approaches. And there is no any consensus. Why not to commit
> working/agreed parts then?

So getting this code into Andrews development tree (as long as he is willing
to accept it) looks very reasonable. We can't change the interface
once we get into the stable kernel because that becomes part of the
ABI.

So all I am saying is that this code is clearly not yet ready for
the stable branch, because we plan to change the sysctl interface.

>> We also need to discuss the system call interface, as without one
>> the functionality is unusable :)
> I also don't see why it can be separated. There is an API in namespaces, and how
> it is mapped into syscalls is another question. At least it doesn't prevent us
> from commiting virtualization itself, agree?

Separating the patches makes a lot of sense. Putting something into
the kernel without any in tree users is a problem.

Eric

Subject: Re: [RFC][PATCH 2/2] Virtualization of IPC
Posted by ebiederm on Fri, 24 Mar 2006 20:09:20 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

> This patch introduces IPC namespaces, which allow to create isolated IPC users
> or containers.
> Introduces CONFIG_IPC_NS and ipc_namespace structure.
> It also uses current->ipc_ns as a pointer to current namespace, which reduces
> places where additional argument to functions should be added.

I don't see where we are freeing the shared memory segments,
the message queues and the semaphores when the last user of the namespace
goes away. Am I missing something?

> --- a/include/linux/ipc.h
> +++ b/include/linux/ipc.h
> @@ -70,6 +70,50 @@ struct kern_ipc_perm
>
> #endif /* __KERNEL__ */
>
> +#include <linux/config.h>
> +
> +#ifdef CONFIG_IPC_NS
> +#include <asm/atomic.h>
> +
> +struct ipc_ids;
> +struct ipc_namespace {
> + atomic_t cnt;
> +
> + struct ipc_ids *sem_ids;
> + int sem_ctls[4];
> + int used_sems;
> +
> + struct ipc_ids *msg_ids;
> + int msg_ctlmax;
> + int msg_ctlmnb;
> + int msg_ctlmni;
> +
> + struct ipc_ids *shm_ids;
> + size_t shm_ctlmax;
> + size_t shm_ctlall;
> + int shm_ctlmni;
> + int shm_total;
> +};

I believe there is a small problem with this implementation.
per namespace counts and limits are fine. But I think we want
to maintain true global limits as well.   I know
concerns of that nature have been expressed in regards
to Daves patch.

> --- a/kernel/fork.c
> +++ b/kernel/fork.c
> @@ -1193,6 +1193,7 @@ static task_t *copy_process(unsigned lon
> attach_pid(p, PIDTYPE_TGID, p->tgid);
> attach_pid(p, PIDTYPE_PID, p->pid);
> get_uts_ns(p->uts_ns);
> + get_ipc_ns(p->ipc_ns);
>
> nr_threads++;
> total_forks++;

Again please move the get outside of the tasklist_lock.

Eric

Subject: Re: [RFC][PATCH 1/2] Virtualization of UTS
Posted by James Morris on Fri, 24 Mar 2006 20:28:09 GMT
View Forum Message <> Reply to Message

On Fri, 24 Mar 2006, Kirill Korotaev wrote:

> This patch introduces utsname namespace in system, which allows to have
> different utsnames on the host.
> Introduces config option CONFIG_UTS_NS and uts_namespace structure for this.
>

Please include patches inline, so they can be quoted inline.

+#define system_utsname (current->uts_ns->name)

This should be a static inline.

+struct uts_namespace *create_uts_ns(void)
+{
+ struct uts_namespace *ns;
+
+ ns = kmalloc(sizeof(struct uts_namespace), GFP_KERNEL);
+ if (ns == NULL)
+ return NULL;
+
+ memset(&ns->name, 0, sizeof(ns->name));
+ atomic_set(&ns->cnt, 1);
+ return ns;
+}

IMHO, it's better to do something like:

```
{
 foo = kmalloc();
 if (foo) {
  stuff;
  etc;
 }
 return foo;
}
```

I suggest you use kzalloc(), too.

Also, I think the API approach needs to be determined before the patches
go into -mm, in case it impacts on the code.

Subject: Re: [RFC][PATCH 2/2] Virtualization of IPC
Posted by Herbert Poetzl on Fri, 24 Mar 2006 21:27:13 GMT
View Forum Message <> Reply to Message

On Fri, Mar 24, 2006 at 11:13:20AM -0800, Dave Hansen wrote:
> On Fri, 2006-03-24 at 20:35 +0300, Kirill Korotaev wrote:
> > This patch introduces IPC namespaces, which allow to create isolated IPC
> > users or containers.
> > Introduces CONFIG_IPC_NS and ipc_namespace structure.
> > It also uses current->ipc_ns as a pointer to current namespace, which
> > reduces places where additional argument to functions should be added.
>
> In three words, I think this has "too many #ifdefs".
>
> The non-containerized or namespaced case should probably just be one,
> static namespace variable that gets wrapped up in some nice #ifdefed
> hlper functions.
>
> For instance, instead of this:
>
> +#ifdef CONFIG_IPC_NS
> +#define msg_ids (*(current->ipc_ns->msg_ids))
> +#endif
>
> Have
>
> #ifdef CONFIG_IPC_NS
> static inline struct ipc_namespace *current_ipc_ns(void)
> {
> return current->ipc_ns;
> }
> #else
> static inline struct ipc_namespace *current_ipc_ns(void)

> {
> return &static_ipc_ns;
> }
> #endif
>
> And use current_ipc_ns()->msg_ids. I can't imagine that gcc can't
> figure that out and turn it back into effectively the same thing.

one issue here, not always 'current' is the right context,
often you handle stuff on behalf of a task, which would
then point to the 'proper' context ...

i.e. something like task_msg_ids(current) is probably
better and more flexible, also I'm still not convinced
that 'per process' is the proper context for those
things, 'per container' or 'per space' would be more
appropriate IMHO ...

more comments to follow, when I got to the patches ...

> I really dislike the idea of replacing nice variables with macros that
> add indirection. They really might fool people. Putting a function
> there is much nicer.
>
> Why avoid to passing these things around as function arguments? Doesn't
> that make it more explicit what is going on, and where the indirection
> is occurring? Does it also make refcounting and lifetime issues easier
> to manage?
>
> BTW, Did you see my version of this?

no, where is it?

maybe we should put all that stuff on a wiki too?

best,
Herbert

>
> -- Dave

Subject: Re: [RFC][PATCH 2/2] Virtualization of IPC
Posted by serue on Mon, 27 Mar 2006 15:33:26 GMT
View Forum Message <> Reply to Message

Quoting Kirill Korotaev (dev@sw.ru):

Thanks for sending the patches.

> +#define msg_ctlmax (current->ipc_ns->msg_ctlmax)
> +#define msg_ctlmnb (current->ipc_ns->msg_ctlmnb)
> +#define msg_ctlmni (current->ipc_ns->msg_ctlmni)

Oh my - woe to the person trying to figure out why

 current->ipc_ns->msg_ctlmax = 20;

isn't compiling.

-serge

Subject: Re: [RFC][PATCH 1/2] Virtualization of UTS
Posted by ebiederm on Mon, 27 Mar 2006 19:40:50 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

>>> This patch introduces utsname namespace in system, which allows to have
>>> different utsnames on the host.
>>> Introduces config option CONFIG_UTS_NS and uts_namespace structure for this.
>> Ok. It looks like we need to resolve the sysctl issues before we merge
>> either patch, into the stable kernel.
> I disagree with you. Right now we can have sysctl and proc for init namespaces
> only.
> And when sysctl and proc are virtualized somehow, we can fix all these.
> I simply don't expect /proc and sysctl to be done quickly. As we have very
> different approaches. And there is no any consensus. Why not to commit
> working/agreed parts then?

So for planning purposes. I don't think we can even if we ignore sysctl
and proc have an implementation that we all agree is stable and safe
before 2.6.17 merge window closes. I do think if we get our act
together something that works and is tested when the 2.6.18 window
opens is very reasonable. (Limited to UTS and sysvipc with other work
waiting until later).

Does that sound like a reasonable and achievable goal?

Eric

Subject: Re: [RFC][PATCH 1/2] Virtualization of UTS
Posted by Sam Vilain on Tue, 28 Mar 2006 03:45:56 GMT

On Fri, 2006-03-24 at 20:31 +0300, Kirill Korotaev wrote:
> +static inline void get_uts_ns(struct uts_namespace *ns)
> +{
> + atomic_inc(&ns->cnt);
> +}
> +
> +static inline void put_uts_ns(struct uts_namespace *ns)
> +{
> + if (atomic_dec_and_test(&ns->cnt))
> + free_uts_ns(ns);
> +}

I think somebody already said this, but this is probably better using kobject as I was asked to for the vx_info. (Documentation/kobject.txt)

Also I think it might be useful to have a count of tasks that refer to the structure, in addition to the count of actual references. In this way you can know whether the resource is "free" before its kobject destructor is called (as the vserver vx_info does).

Perhaps that abstraction is best to put in when it becomes "useful", like you have a situation where you want to do something when the last process with a utsname exits, but before the last kthread referencing the structure stops (eg, a sleeping process reading /proc somewhere).

Otherwise, nice and simple; I could quite easily at this point plug this into the syscall infrastructure I posted earlier (once it is reworked based on people's comments), and provide tests for this.

Sam.

Subject: Re: [RFC][PATCH 2/2] Virtualization of IPC
Posted by Sam Vilain on Tue, 28 Mar 2006 05:26:23 GMT

On Fri, 2006-03-24 at 22:27 +0100, Herbert Poetzl wrote:
> maybe we should put all that stuff on a wiki too?

Good heavens, for the love of life and everything pure and good in the world, no!

Perhaps we can get a kernel.org git tree? Each submission can be put in as seperate branches.

For instance I have already been doing this for several submissions, see

http://vserver.utsl.gen.nz/gitweb/?p=vserver;a=heads

This doesn't include Dave's or Kirill's patches yet (as of -0328T15:53 +12).

OK, now it does (as of -0328T17:23+12).

I am more than happy to continue to track people's submissions in this way, but they probably belong on kernel.org

Sam.
