Subject: [PATCH] IPC namespace
Posted by Kirill Korotaev on Fri, 09 Jun 2006 14:55:48 GMT
View Forum Message <> Reply to Message

The patches in this thread add IPC namespace functionality
additionally to already included in -mm tree UTS namespace.

This patch set allows to unshare IPCs and have a private set
of IPC objects (sem, shm, msg) inside namespace. Basically, it is
another building block of containers functionality.

Tested with LTP inside namespaces.

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

P.S. patches are against linux-2.6.17-rc6-mm1

Subject: [PATCH 1/6] IPC namespace core
Posted by Kirill Korotaev on Fri, 09 Jun 2006 15:01:38 GMT
View Forum Message <> Reply to Message

This patch implements core IPC namespace changes:
- ipc_namespace structure
- new config option CONFIG_IPC_NS
- adds CLONE_NEWIPC flag
- unshare support

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

```
--- ./include/linux/init_task.h.ipcns 2006-06-06 14:47:58.000000000 +0400
+++ ./include/linux/init_task.h 2006-06-08 14:28:23.000000000 +0400
@@ -73,6 +73,7 @@ extern struct nsproxy init_nsproxy;
  .count  = ATOMIC_INIT(1),    \
  .nslock  = SPIN_LOCK_UNLOCKED,    \
  .uts_ns  = &init_uts_ns,     \
+ .ipc_ns  = &init_ipc_ns,     \
  .namespace = NULL,      \
 }

--- ./include/linux/ipc.h.ipcns 2006-04-21 11:59:36.000000000 +0400
+++ ./include/linux/ipc.h 2006-06-08 15:43:43.000000000 +0400
@@ -2,6 +2,7 @@
 #define _LINUX_IPC_H
```

```
 #include <linux/types.h>
+#include <linux/kref.h>

 #define IPC_PRIVATE ((__kernel_key_t) 0)

@@ -68,6 +69,41 @@ struct kern_ipc_perm
  void  *security;
 };

+struct ipc_ids;
+struct ipc_namespace {
+ struct kref kref;
+ struct ipc_ids *ids[3];
+
+ int  sem_ctls[4];
+ int  used_sems;
+
+ int  msg_ctlmax;
+ int  msg_ctlmnb;
+ int  msg_ctlmni;
+
+ size_t  shm_ctlmax;
+ size_t  shm_ctlall;
+ int  shm_ctlmni;
+ int  shm_tot;
+};
+
+extern struct ipc_namespace init_ipc_ns;
+extern void free_ipc_ns(struct kref *kref);
+extern int copy_ipcs(unsigned long flags, struct task_struct *tsk);
+extern int unshare_ipcs(unsigned long flags, struct ipc_namespace **ns);
+
+static inline struct ipc_namespace *get_ipc_ns(struct ipc_namespace *ns)
+{
+ if (ns)
+  kref_get(&ns->kref);
+ return ns;
+}
+
+static inline void put_ipc_ns(struct ipc_namespace *ns)
+{
+ kref_put(&ns->kref, free_ipc_ns);
+}
+
 #endif /* __KERNEL__ */

 #endif /* _LINUX_IPC_H */
--- ./include/linux/nsproxy.h.ipcns 2006-06-06 14:47:58.000000000 +0400
```

```
+++ ./include/linux/nsproxy.h 2006-06-08 15:28:02.000000000 +0400
@@ -6,6 +6,7 @@

 struct namespace;
 struct uts_namespace;
+struct ipc_namespace;

 /*
  * A structure to contain pointers to all per-process
@@ -23,6 +24,7 @@ struct nsproxy {
  atomic_t count;
  spinlock_t nslock;
  struct uts_namespace *uts_ns;
+ struct ipc_namespace *ipc_ns;
  struct namespace *namespace;
 };
 extern struct nsproxy init_nsproxy;
--- ./include/linux/sched.h.ipcns 2006-06-06 14:47:58.000000000 +0400
+++ ./include/linux/sched.h 2006-06-08 14:28:23.000000000 +0400
@@ -25,6 +25,7 @@
 #define CLONE_CHILD_SETTID 0x01000000 /* set the TID in the child */
 #define CLONE_STOPPED  0x02000000 /* Start in stopped state */
 #define CLONE_NEWUTS  0x04000000 /* New utsname group? */
+#define CLONE_NEWIPC  0x08000000 /* New ipcs */

 /*
  * Scheduling policies
--- ./init/Kconfig.ipcns 2006-06-06 14:47:58.000000000 +0400
+++ ./init/Kconfig 2006-06-09 14:18:09.000000000 +0400
@@ -137,6 +137,15 @@ config SYSVIPC
    section 6.4 of the Linux Programmer's Guide, available from
    <http://www.tldp.org/guides.html>.

+config IPC_NS
+ bool "IPC Namespaces"
+ depends on SYSVIPC
+ default n
+ help
+   Support ipc namespaces.  This allows containers, i.e. virtual
+   environments, to use ipc namespaces to provide different ipc
+   objects for different servers.  If unsure, say N.
+
 config POSIX_MQUEUE
  bool "POSIX Message Queues"
  depends on NET && EXPERIMENTAL
--- ./kernel/fork.c.ipcns 2006-06-06 14:47:58.000000000 +0400
+++ ./kernel/fork.c 2006-06-08 15:31:03.000000000 +0400
@@ -1592,6 +1592,7 @@ asmlinkage long sys_unshare(unsigned lon
```

```
  struct sem_undo_list *new_ulist = NULL;
  struct nsproxy *new_nsproxy = NULL, *old_nsproxy = NULL;
  struct uts_namespace *uts, *new_uts = NULL;
+ struct ipc_namespace *ipc, *new_ipc = NULL;

  check_unshare_flags(&unshare_flags);

@@ -1617,18 +1618,20 @@ asmlinkage long sys_unshare(unsigned lon
   goto bad_unshare_cleanup_fd;
  if ((err = unshare_utsname(unshare_flags, &new_uts)))
   goto bad_unshare_cleanup_semundo;
+ if ((err = unshare_ipcs(unshare_flags, &new_ipc)))
+  goto bad_unshare_cleanup_uts;

  if (new_ns || new_uts) {
   old_nsproxy = current->nsproxy;
   new_nsproxy = dup_namespaces(old_nsproxy);
   if (!new_nsproxy) {
    err = -ENOMEM;
-   goto bad_unshare_cleanup_uts;
+   goto bad_unshare_cleanup_ipc;
   }
  }

  if (new_fs || new_ns || new_sigh || new_mm || new_fd || new_ulist ||
-    new_uts) {
+    new_uts || new_ipc) {

   task_lock(current);

@@ -1676,12 +1679,22 @@ asmlinkage long sys_unshare(unsigned lon
    new_uts = uts;
   }

+  if (new_ipc) {
+   ipc = current->nsproxy->ipc_ns;
+   current->nsproxy->ipc_ns = new_ipc;
+   new_ipc = ipc;
+  }
+
   task_unlock(current);
  }

  if (new_nsproxy)
   put_nsproxy(new_nsproxy);

+bad_unshare_cleanup_ipc:
+ if (new_ipc)
```

```
+  put_ipc_ns(new_ipc);
+
 bad_unshare_cleanup_uts:
  if (new_uts)
   put_uts_ns(new_uts);
--- ./kernel/nsproxy.c.ipcns 2006-06-06 14:47:59.000000000 +0400
+++ ./kernel/nsproxy.c 2006-06-09 14:22:31.000000000 +0400
@@ -7,6 +7,10 @@
  *  modify it under the terms of the GNU General Public License as
  *  published by the Free Software Foundation, version 2 of the
  *  License.
+ *
+ *  Jun 2006 - namespaces support
+ *             OpenVZ, SWsoft Inc.
+ *             Pavel Emelianov <xemul@openvz.org>
  */

 #include <linux/module.h>
@@ -59,6 +63,8 @@ struct nsproxy *dup_namespaces(struct ns
    get_namespace(ns->namespace);
   if (ns->uts_ns)
    get_uts_ns(ns->uts_ns);
+  if (ns->ipc_ns)
+   get_ipc_ns(ns->ipc_ns);
  }

  return ns;
@@ -79,7 +85,7 @@ int copy_namespaces(int flags, struct ta

  get_nsproxy(old_ns);

- if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS)))
+ if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC)))
   return 0;

  new_ns = clone_namespaces(old_ns);
@@ -91,24 +97,31 @@ int copy_namespaces(int flags, struct ta
  tsk->nsproxy = new_ns;

  err = copy_namespace(flags, tsk);
- if (err) {
-  tsk->nsproxy = old_ns;
-  put_nsproxy(new_ns);
-  goto out;
- }
+ if (err)
+  goto out_ns;
```

```
  err = copy_utsname(flags, tsk);
- if (err) {
-  if (new_ns->namespace)
-   put_namespace(new_ns->namespace);
-  tsk->nsproxy = old_ns;
-  put_nsproxy(new_ns);
-  goto out;
- }
+ if (err)
+  goto out_uts;
+
+ err = copy_ipcs(flags, tsk);
+ if (err)
+  goto out_ipc;

 out:
  put_nsproxy(old_ns);
  return err;
+
+out_ipc:
+ if (new_ns->uts_ns)
+  put_uts_ns(new_ns->uts_ns);
+out_uts:
+ if (new_ns->namespace)
+  put_namespace(new_ns->namespace);
+out_ns:
+ tsk->nsproxy = old_ns;
+ put_nsproxy(new_ns);
+ goto out;
 }

 void free_nsproxy(struct nsproxy *ns)
@@ -117,5 +130,7 @@ void free_nsproxy(struct nsproxy *ns)
    put_namespace(ns->namespace);
   if (ns->uts_ns)
    put_uts_ns(ns->uts_ns);
+  if (ns->ipc_ns)
+   put_ipc_ns(ns->ipc_ns);
   kfree(ns);
 }
```

Subject: [PATCH 2/6] IPC namespace - utils
Posted by Kirill Korotaev on Fri, 09 Jun 2006 15:05:29 GMT
View Forum Message <> Reply to Message

This patch adds basic IPC namespace functionality to
IPC utils:

```
- init_ipc_ns
- copy/clone/unshare/free IPC ns
- /proc preparations

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

--- ./ipc/util.c.ipcns 2006-05-18 12:05:15.000000000 +0400
+++ ./ipc/util.c 2006-06-09 14:19:30.000000000 +0400
@@ -12,6 +12,9 @@
  *          Mingming Cao <cmm@us.ibm.com>
  * Mar 2006 - support for audit of ipc object properties
  *          Dustin Kirkland <dustin.kirkland@us.ibm.com>
+ * Jun 2006 - namespaces ssupport
+ *          OpenVZ, SWsoft Inc.
+ *          Pavel Emelianov <xemul@openvz.org>
  */

 #include <linux/config.h>
@@ -30,6 +33,7 @@
 #include <linux/seq_file.h>
 #include <linux/proc_fs.h>
 #include <linux/audit.h>
+#include <linux/nsproxy.h>

 #include <asm/unistd.h>

@@ -38,10 +42,126 @@
 struct ipc_proc_iface {
  const char *path;
  const char *header;
- struct ipc_ids *ids;
+ int ids;
  int (*show)(struct seq_file *, void *);
 };

+struct ipc_namespace init_ipc_ns = {
+ .kref = {
+  .refcount = ATOMIC_INIT(2),
+ },
+};
+
+#ifdef CONFIG_IPC_NS
+static struct ipc_namespace *clone_ipc_ns(struct ipc_namespace *old_ns)
+{
+ int err;
+ struct ipc_namespace *ns;
+
```

```
+ err = -ENOMEM;
+ ns = kmalloc(sizeof(struct ipc_namespace), GFP_KERNEL);
+ if (ns == NULL)
+  goto err_mem;
+
+ err = sem_init_ns(ns);
+ if (err)
+  goto err_sem;
+ err = msg_init_ns(ns);
+ if (err)
+  goto err_msg;
+ err = shm_init_ns(ns);
+ if (err)
+  goto err_shm;
+
+ kref_init(&ns->kref);
+ return ns;
+
+err_shm:
+ msg_exit_ns(ns);
+err_msg:
+ sem_exit_ns(ns);
+err_sem:
+ kfree(ns);
+err_mem:
+ return ERR_PTR(err);
+}
+
+int unshare_ipcs(unsigned long unshare_flags, struct ipc_namespace **new_ipc)
+{
+ struct ipc_namespace *new;
+
+ if (unshare_flags & CLONE_NEWIPC) {
+  if (!capable(CAP_SYS_ADMIN))
+   return -EPERM;
+
+  new = clone_ipc_ns(current->nsproxy->ipc_ns);
+  if (IS_ERR(new))
+   return PTR_ERR(new);
+
+  *new_ipc = new;
+ }
+
+ return 0;
+}
+
+int copy_ipcs(unsigned long flags, struct task_struct *tsk)
+{
```

```
+ struct ipc_namespace *old_ns = tsk->nsproxy->ipc_ns;
+ struct ipc_namespace *new_ns;
+ int err = 0;
+
+ if (!old_ns)
+  return 0;
+
+ get_ipc_ns(old_ns);
+
+ if (!(flags & CLONE_NEWIPC))
+  return 0;
+
+ if (!capable(CAP_SYS_ADMIN)) {
+  err = -EPERM;
+  goto out;
+ }
+
+ new_ns = clone_ipc_ns(old_ns);
+ if (!new_ns) {
+  err = -ENOMEM;
+  goto out;
+ }
+
+ tsk->nsproxy->ipc_ns = new_ns;
+out:
+ put_ipc_ns(old_ns);
+ return err;
+}
+
+void free_ipc_ns(struct kref *kref)
+{
+ struct ipc_namespace *ns;
+
+ ns = container_of(kref, struct ipc_namespace, kref);
+ sem_exit_ns(ns);
+ msg_exit_ns(ns);
+ shm_exit_ns(ns);
+ kfree(ns);
+}
+#else
+int unshare_ipcs(unsigned long flags, struct ipc_namespace **ns)
+{
+ return -EINVAL;
+}
+
+int copy_ipcs(unsigned long flags, struct task_struct *tsk)
+{
+ return 0;
```

```
+}
+
+void free_ipc_ns(struct kref *kref)
+{
+ BUG(); /* init_ipc_ns should never be put */
+}
+#endif
+
 /**
  * ipc_init - initialise IPC subsystem
  *
@@ -68,7 +188,7 @@ __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;

@@ -111,8 +231,7 @@ static struct file_operations sysvipc_pr
  * @show: show routine.
  */
 void __init ipc_init_proc_interface(const char *path, const char *header,
-        struct ipc_ids *ids,
-        int (*show)(struct seq_file *, void *))
+  int ids, int (*show)(struct seq_file *, void *))
 {
  struct proc_dir_entry *pde;
  struct ipc_proc_iface *iface;
@@ -636,6 +755,9 @@ static void *sysvipc_proc_next(struct se
  struct ipc_proc_iface *iface = s->private;
  struct kern_ipc_perm *ipc = it;
  loff_t p;
+ struct ipc_ids *ids;
+
+ ids = current->nsproxy->ipc_ns->ids[iface->ids];

  /* If we had an ipc id locked before, unlock it */
  if (ipc && ipc != SEQ_START_TOKEN)
@@ -645,8 +767,8 @@ static void *sysvipc_proc_next(struct se
   * p = *pos - 1 (because id 0 starts at position 1)
   *          + 1 (because we increment the position by one)
   */
- for (p = *pos; p <= iface->ids->max_id; p++) {
-  if ((ipc = ipc_lock(iface->ids, p)) != NULL) {
+ for (p = *pos; p <= ids->max_id; p++) {
+  if ((ipc = ipc_lock(ids, p)) != NULL) {
```

```
    *pos = p + 1;
    return ipc;
   }
@@ -665,12 +787,15 @@ static void *sysvipc_proc_start(struct s
  struct ipc_proc_iface *iface = s->private;
  struct kern_ipc_perm *ipc;
  loff_t p;
+ struct ipc_ids *ids;
+
+ ids = current->nsproxy->ipc_ns->ids[iface->ids];

  /*
   * Take the lock - this will be released by the corresponding
   * call to stop().
   */
- mutex_lock(&iface->ids->mutex);
+ mutex_lock(&ids->mutex);

  /* pos < 0 is invalid */
  if (*pos < 0)
@@ -681,8 +806,8 @@ static void *sysvipc_proc_start(struct s
   return SEQ_START_TOKEN;

  /* Find the (pos-1)th ipc */
- for (p = *pos - 1; p <= iface->ids->max_id; p++) {
-  if ((ipc = ipc_lock(iface->ids, p)) != NULL) {
+ for (p = *pos - 1; p <= ids->max_id; p++) {
+  if ((ipc = ipc_lock(ids, p)) != NULL) {
    *pos = p + 1;
    return ipc;
   }
@@ -694,13 +819,15 @@ static void sysvipc_proc_stop(struct seq
 {
  struct kern_ipc_perm *ipc = it;
  struct ipc_proc_iface *iface = s->private;
+ struct ipc_ids *ids;

  /* If we had a locked segment, release it */
  if (ipc && ipc != SEQ_START_TOKEN)
   ipc_unlock(ipc);

+ ids = current->nsproxy->ipc_ns->ids[iface->ids];
  /* Release the lock we took in start() */
- mutex_unlock(&iface->ids->mutex);
+ mutex_unlock(&ids->mutex);
 }

 static int sysvipc_proc_show(struct seq_file *s, void *it)
```

```
--- ./ipc/util.h.ipcns 2006-04-21 11:59:36.000000000 +0400
+++ ./ipc/util.h 2006-06-09 14:24:40.000000000 +0400
@@ -3,6 +3,8 @@
  * Copyright (C) 1999 Christoph Rohland
  *
  * ipc helper functions (c) 1999 Manfred Spraul <manfred@colorfullife.com>
+ * namespaces support.      2006 OpenVZ, SWsoft Inc.
+ *                          Pavel Emelianov <xemul@openvz.org>
  */

 #ifndef _IPC_UTIL_H
@@ -15,6 +17,14 @@ void sem_init (void);
 void msg_init (void);
 void shm_init (void);

+int sem_init_ns(struct ipc_namespace *ns);
+int msg_init_ns(struct ipc_namespace *ns);
+int shm_init_ns(struct ipc_namespace *ns);
+
+void sem_exit_ns(struct ipc_namespace *ns);
+void msg_exit_ns(struct ipc_namespace *ns);
+void shm_exit_ns(struct ipc_namespace *ns);
+
 struct ipc_id_ary {
  int size;
  struct kern_ipc_perm *p[0];
@@ -31,15 +41,23 @@ struct ipc_ids {
 };

 struct seq_file;
-void __init ipc_init_ids(struct ipc_ids* ids, int size);
+#ifdef CONFIG_IPC_NS
+#define __ipc_init
+#else
+#define __ipc_init __init
+#endif
+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,
-        int (*show)(struct seq_file *, void *));
+  int ids, int (*show)(struct seq_file *, void *));
 #else
 #define ipc_init_proc_interface(path, header, ids, show) do {} while (0)
 #endif

+#define IPC_SEM_IDS 0
+#define IPC_MSG_IDS 1
```

```
+#define IPC_SHM_IDS 2
+
 /* must be called with ids->mutex acquired.*/
 int ipc_findkey(struct ipc_ids* ids, key_t key);
 int ipc_addid(struct ipc_ids* ids, struct kern_ipc_perm* new, int size);
```

Subject: [PATCH 3/6] IPC namespace - msg
Posted by Kirill Korotaev on Fri, 09 Jun 2006 15:07:05 GMT
View Forum Message <> Reply to Message

IPC namespace support for IPC msg code.

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

```
--- ./ipc/msg.c.ipcns 2006-06-06 14:47:58.000000000 +0400
+++ ./ipc/msg.c 2006-06-09 14:20:57.000000000 +0400
@@ -16,6 +16,10 @@
  *
  * support for audit of ipc object properties and permission changes
  * Dustin Kirkland <dustin.kirkland@us.ibm.com>
+ *
+ * namespaces support
+ * OpenVZ, SWsoft Inc.
+ * Pavel Emelianov <xemul@openvz.org>
  */

 #include <linux/capability.h>
@@ -32,16 +36,12 @@
 #include <linux/audit.h>
 #include <linux/seq_file.h>
 #include <linux/mutex.h>
+#include <linux/nsproxy.h>

 #include <asm/current.h>
 #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;
```

```
@@ -68,32 +68,75 @@ 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;
+static struct ipc_ids init_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))
-#define msg_checkid(msq, msgid) \
- ipc_checkid(&msg_ids,&msq->q_perm,msgid)
-#define msg_buildid(id, seq) \
- ipc_buildid(&msg_ids, id, seq)
+#define msg_ids(ns) (*((ns)->ids[IPC_MSG_IDS]))

-static void freeque (struct msg_queue *msq, int id);
-static int newque (key_t key, int msgflg);
+#define msg_lock(ns, id) ((struct msg_queue*)ipc_lock(&msg_ids(ns), id))
+#define msg_unlock(msq)  ipc_unlock(&(msq)->q_perm)
+#define msg_rmid(ns, id) ((struct msg_queue*)ipc_rmid(&msg_ids(ns),id))
+#define msg_checkid(ns, msq, msgid) \
+ ipc_checkid(&msg_ids(ns),&msq->q_perm,msgid)
+#define msg_buildid(ns, id, seq) \
+ ipc_buildid(&msg_ids(ns), id, seq)
+
+static void freeque (struct ipc_namespace *ns, struct msg_queue *msq, int id);
+static int newque (struct ipc_namespace *ns, key_t key, int msgflg);
 #ifdef CONFIG_PROC_FS
 static int sysvipc_msg_proc_show(struct seq_file *s, void *it);
 #endif

+static void __ipc_init __msg_init_ns(struct ipc_namespace *ns, struct ipc_ids *ids)
+{
+ ns->ids[IPC_MSG_IDS] = ids;
+ ns->msg_ctlmax = MSGMAX;
+ ns->msg_ctlmnb = MSGMNB;
+ ns->msg_ctlmni = MSGMNI;
+ ipc_init_ids(ids, ns->msg_ctlmni);
+}
+
+#ifdef CONFIG_IPC_NS
+int msg_init_ns(struct ipc_namespace *ns)
+{
+ struct ipc_ids *ids;
+
+ ids = kmalloc(sizeof(struct ipc_ids), GFP_KERNEL);
+ if (ids == NULL)
```

```
+  return -ENOMEM;
+
+ __msg_init_ns(ns, ids);
+ return 0;
+}
+
+void msg_exit_ns(struct ipc_namespace *ns)
+{
+ int i;
+ struct msg_queue *msq;
+
+ mutex_lock(&msg_ids(ns).mutex);
+ for (i = 0; i <= msg_ids(ns).max_id; i++) {
+  msq = msg_lock(ns, i);
+  if (msq == NULL)
+   continue;
+
+  freeque(ns, msq, i);
+ }
+ mutex_unlock(&msg_ids(ns).mutex);
+
+ kfree(ns->ids[IPC_MSG_IDS]);
+ ns->ids[IPC_MSG_IDS] = NULL;
+}
+#endif
+
 void __init msg_init (void)
 {
- ipc_init_ids(&msg_ids,msg_ctlmni);
+ __msg_init_ns(&init_ipc_ns, &init_msg_ids);
  ipc_init_proc_interface("sysvipc/msg",
     "       key      msqid perms      cbytes       qnum lspid lrpid   uid   gid  cuid  cgid      stime      rtime
     ctime\n",
-    &msg_ids,
-    sysvipc_msg_proc_show);
+    IPC_MSG_IDS, sysvipc_msg_proc_show);
 }

-static int newque (key_t key, int msgflg)
+static int newque (struct ipc_namespace *ns, key_t key, int msgflg)
 {
  int id;
  int retval;
@@ -113,18 +156,18 @@ static int newque (key_t key, int msgflg
   return retval;
  }

- id = ipc_addid(&msg_ids, &msq->q_perm, msg_ctlmni);
```

```
+ id = ipc_addid(&msg_ids(ns), &msq->q_perm, ns->msg_ctlmni);
  if(id == -1) {
   security_msg_queue_free(msq);
   ipc_rcu_putref(msq);
   return -ENOSPC;
  }

- msq->q_id = msg_buildid(id,msq->q_perm.seq);
+ msq->q_id = msg_buildid(ns, id, msq->q_perm.seq);
  msq->q_stime = msq->q_rtime = 0;
  msq->q_ctime = get_seconds();
  msq->q_cbytes = msq->q_qnum = 0;
- msq->q_qbytes = msg_ctlmnb;
+ msq->q_qbytes = ns->msg_ctlmnb;
  msq->q_lspid = msq->q_lrpid = 0;
  INIT_LIST_HEAD(&msq->q_messages);
  INIT_LIST_HEAD(&msq->q_receivers);
@@ -187,13 +230,13 @@ static void expunge_all(struct msg_queue
  * msg_ids.mutex and the spinlock for this message queue is hold
  * before freeque() is called. msg_ids.mutex remains locked on exit.
  */
-static void freeque (struct msg_queue *msq, int id)
+static void freeque (struct ipc_namespace *ns, struct msg_queue *msq, int id)
 {
  struct list_head *tmp;

  expunge_all(msq,-EIDRM);
  ss_wakeup(&msq->q_senders,1);
- msq = msg_rmid(id);
+ msq = msg_rmid(ns, id);
  msg_unlock(msq);

  tmp = msq->q_messages.next;
@@ -212,31 +255,34 @@ asmlinkage long sys_msgget (key_t key, i
 {
  int id, ret = -EPERM;
  struct msg_queue *msq;
+ struct ipc_namespace *ns;
+
+ ns = current->nsproxy->ipc_ns;

- mutex_lock(&msg_ids.mutex);
+ mutex_lock(&msg_ids(ns).mutex);
  if (key == IPC_PRIVATE)
-  ret = newque(key, msgflg);
- else if ((id = ipc_findkey(&msg_ids, key)) == -1) { /* key not used */
+  ret = newque(ns, key, msgflg);
+ else if ((id = ipc_findkey(&msg_ids(ns), key)) == -1) { /* key not used */
```

```
   if (!(msgflg & IPC_CREAT))
    ret = -ENOENT;
   else
-   ret = newque(key, msgflg);
+   ret = newque(ns, key, msgflg);
  } else if (msgflg & IPC_CREAT && msgflg & IPC_EXCL) {
   ret = -EEXIST;
  } else {
-  msq = msg_lock(id);
+  msq = msg_lock(ns, id);
   BUG_ON(msq==NULL);
   if (ipcperms(&msq->q_perm, msgflg))
    ret = -EACCES;
   else {
-   int qid = msg_buildid(id, msq->q_perm.seq);
+   int qid = msg_buildid(ns, id, msq->q_perm.seq);
       ret = security_msg_queue_associate(msq, msgflg);
    if (!ret)
     ret = qid;
   }
   msg_unlock(msq);
  }
- mutex_unlock(&msg_ids.mutex);
+ mutex_unlock(&msg_ids(ns).mutex);
  return ret;
 }

@@ -337,11 +383,13 @@ asmlinkage long sys_msgctl (int msqid, i
  struct msg_queue *msq;
  struct msq_setbuf setbuf;
  struct kern_ipc_perm *ipcp;
+ struct ipc_namespace *ns;

  if (msqid < 0 || cmd < 0)
   return -EINVAL;

  version = ipc_parse_version(&cmd);
+ ns = current->nsproxy->ipc_ns;

  switch (cmd) {
  case IPC_INFO:
@@ -361,14 +409,14 @@ asmlinkage long sys_msgctl (int msqid, i
    return err;

   memset(&msginfo,0,sizeof(msginfo));
-  msginfo.msgmni = msg_ctlmni;
-  msginfo.msgmax = msg_ctlmax;
-  msginfo.msgmnb = msg_ctlmnb;
```

```
+  msginfo.msgmni = ns->msg_ctlmni;
+  msginfo.msgmax = ns->msg_ctlmax;
+  msginfo.msgmnb = ns->msg_ctlmnb;
   msginfo.msgssz = MSGSSZ;
   msginfo.msgseg = MSGSEG;
-  mutex_lock(&msg_ids.mutex);
+  mutex_lock(&msg_ids(ns).mutex);
   if (cmd == MSG_INFO) {
-   msginfo.msgpool = msg_ids.in_use;
+   msginfo.msgpool = msg_ids(ns).in_use;
    msginfo.msgmap = atomic_read(&msg_hdrs);
    msginfo.msgtql = atomic_read(&msg_bytes);
   } else {
@@ -376,8 +424,8 @@ asmlinkage long sys_msgctl (int msqid, i
    msginfo.msgpool = MSGPOOL;
    msginfo.msgtql = MSGTQL;
   }
-  max_id = msg_ids.max_id;
-  mutex_unlock(&msg_ids.mutex);
+  max_id = msg_ids(ns).max_id;
+  mutex_unlock(&msg_ids(ns).mutex);
   if (copy_to_user (buf, &msginfo, sizeof(struct msginfo)))
    return -EFAULT;
   return (max_id < 0) ? 0: max_id;
@@ -389,20 +437,20 @@ asmlinkage long sys_msgctl (int msqid, i
   int success_return;
   if (!buf)
    return -EFAULT;
-  if(cmd == MSG_STAT && msqid >= msg_ids.entries->size)
+  if(cmd == MSG_STAT && msqid >= msg_ids(ns).entries->size)
    return -EINVAL;

   memset(&tbuf,0,sizeof(tbuf));

-  msq = msg_lock(msqid);
+  msq = msg_lock(ns, msqid);
   if (msq == NULL)
    return -EINVAL;

   if(cmd == MSG_STAT) {
-   success_return = msg_buildid(msqid, msq->q_perm.seq);
+   success_return = msg_buildid(ns, msqid, msq->q_perm.seq);
   } else {
    err = -EIDRM;
-   if (msg_checkid(msq,msqid))
+   if (msg_checkid(ns, msq,msqid))
     goto out_unlock;
    success_return = 0;
```

```
   }
@@ -440,14 +488,14 @@ asmlinkage long sys_msgctl (int msqid, i
    return  -EINVAL;
   }

- mutex_lock(&msg_ids.mutex);
- msq = msg_lock(msqid);
+ mutex_lock(&msg_ids(ns).mutex);
+ msq = msg_lock(ns, msqid);
   err=-EINVAL;
   if (msq == NULL)
    goto out_up;

   err = -EIDRM;
- if (msg_checkid(msq,msqid))
+ if (msg_checkid(ns, msq,msqid))
    goto out_unlock_up;
   ipcp = &msq->q_perm;

@@ -474,7 +522,7 @@ asmlinkage long sys_msgctl (int msqid, i
   case IPC_SET:
   {
    err = -EPERM;
-  if (setbuf.qbytes > msg_ctlmnb && !capable(CAP_SYS_RESOURCE))
+  if (setbuf.qbytes > ns->msg_ctlmnb && !capable(CAP_SYS_RESOURCE))
     goto out_unlock_up;

    msq->q_qbytes = setbuf.qbytes;
@@ -496,12 +544,12 @@ asmlinkage long sys_msgctl (int msqid, i
    break;
   }
   case IPC_RMID:
-  freeque (msq, msqid);
+  freeque (ns, msq, msqid);
    break;
   }
   err = 0;
  out_up:
- mutex_unlock(&msg_ids.mutex);
+ mutex_unlock(&msg_ids(ns).mutex);
   return err;
  out_unlock_up:
   msg_unlock(msq);
@@ -570,8 +618,11 @@ asmlinkage long sys_msgsnd (int msqid, s
   struct msg_msg *msg;
   long mtype;
   int err;
+ struct ipc_namespace *ns;
```

```
+
+ ns = current->nsproxy->ipc_ns;

- if (msgsz > msg_ctlmax || (long) msgsz < 0 || msqid < 0)
+ if (msgsz > ns->msg_ctlmax || (long) msgsz < 0 || msqid < 0)
   return -EINVAL;
  if (get_user(mtype, &msgp->mtype))
   return -EFAULT;
@@ -585,13 +636,13 @@ asmlinkage long sys_msgsnd (int msqid, s
  msg->m_type = mtype;
  msg->m_ts = msgsz;

- msq = msg_lock(msqid);
+ msq = msg_lock(ns, msqid);
  err=-EINVAL;
  if(msq==NULL)
   goto out_free;

  err= -EIDRM;
- if (msg_checkid(msq,msqid))
+ if (msg_checkid(ns, msq,msqid))
   goto out_unlock_free;

  for (;;) {
@@ -682,17 +733,19 @@ asmlinkage long sys_msgrcv (int msqid, s
  struct msg_queue *msq;
  struct msg_msg *msg;
  int mode;
+ struct ipc_namespace *ns;

  if (msqid < 0 || (long) msgsz < 0)
   return -EINVAL;
  mode = convert_mode(&msgtyp,msgflg);
+ ns = current->nsproxy->ipc_ns;

- msq = msg_lock(msqid);
+ msq = msg_lock(ns, msqid);
  if(msq==NULL)
   return -EINVAL;

  msg = ERR_PTR(-EIDRM);
- if (msg_checkid(msq,msqid))
+ if (msg_checkid(ns, msq, msqid))
   goto out_unlock;

  for (;;) {
```

Subject: [PATCH 4/6] IPC namespace - sem
Posted by Kirill Korotaev on Fri, 09 Jun 2006 15:08:33 GMT
View Forum Message <> Reply to Message

IPC namespace support for IPC sem code.

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

```
--- ./ipc/sem.c.ipcns 2006-06-06 14:47:58.000000000 +0400
+++ ./ipc/sem.c 2006-06-09 14:20:06.000000000 +0400
@@ -64,6 +64,10 @@
  *
  * support for audit of ipc object properties and permission changes
  * Dustin Kirkland <dustin.kirkland@us.ibm.com>
+ *
+ * namespaces support
+ * OpenVZ, SWsoft Inc.
+ * Pavel Emelianov <xemul@openvz.org>
  */

 #include <linux/config.h>
@@ -79,22 +83,25 @@
 #include <linux/capability.h>
 #include <linux/seq_file.h>
 #include <linux/mutex.h>
+#include <linux/nsproxy.h>

 #include <asm/uaccess.h>
 #include "util.h"

+#define sem_ids(ns) (*((ns)->ids[IPC_SEM_IDS]))

-#define sem_lock(id) ((struct sem_array*)ipc_lock(&sem_ids,id))
-#define sem_unlock(sma) ipc_unlock(&(sma)->sem_perm)
-#define sem_rmid(id) ((struct sem_array*)ipc_rmid(&sem_ids,id))
-#define sem_checkid(sma, semid) \
- 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;
+#define sem_lock(ns, id) ((struct sem_array*)ipc_lock(&sem_ids(ns), id))
+#define sem_unlock(sma)  ipc_unlock(&(sma)->sem_perm)
+#define sem_rmid(ns, id) ((struct sem_array*)ipc_rmid(&sem_ids(ns), id))
+#define sem_checkid(ns, sma, semid) \
+ ipc_checkid(&sem_ids(ns),&sma->sem_perm,semid)
+#define sem_buildid(ns, id, seq) \
+ ipc_buildid(&sem_ids(ns), id, seq)
```

```
-static int newary (key_t, int, int);
-static void freeary (struct sem_array *sma, int id);
+static struct ipc_ids init_sem_ids;
+
+static int newary (struct ipc_namespace *, key_t, int, int);
+static void freeary (struct ipc_namespace *ns, struct sem_array *sma, int id);
 #ifdef CONFIG_PROC_FS
 static int sysvipc_sem_proc_show(struct seq_file *s, void *it);
 #endif
@@ -111,22 +118,61 @@ static int sysvipc_sem_proc_show(struct
  *
  */

-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])
+#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;
+static void __ipc_init __sem_init_ns(struct ipc_namespace *ns, struct ipc_ids *ids)
+{
+ ns->ids[IPC_SEM_IDS] = ids;
+ ns->sc_semmsl = SEMMSL;
+ ns->sc_semmns = SEMMNS;
+ ns->sc_semopm = SEMOPM;
+ ns->sc_semmni = SEMMNI;
+ ns->used_sems = 0;
+ ipc_init_ids(ids, ns->sc_semmni);
+}
+
+#ifdef CONFIG_IPC_NS
+int sem_init_ns(struct ipc_namespace *ns)
+{
+ struct ipc_ids *ids;
+
+ ids = kmalloc(sizeof(struct ipc_ids), GFP_KERNEL);
+ if (ids == NULL)
+  return -ENOMEM;
+
+ __sem_init_ns(ns, ids);
+ return 0;
+}
+
```

```
+void sem_exit_ns(struct ipc_namespace *ns)
+{
+ int i;
+ struct sem_array *sma;
+
+ mutex_lock(&sem_ids(ns).mutex);
+ for (i = 0; i <= sem_ids(ns).max_id; i++) {
+  sma = sem_lock(ns, i);
+  if (sma == NULL)
+   continue;
+
+  freeary(ns, sma, i);
+ }
+ mutex_unlock(&sem_ids(ns).mutex);
+
+ kfree(ns->ids[IPC_SEM_IDS]);
+ ns->ids[IPC_SEM_IDS] = NULL;
+}
+#endif

 void __init sem_init (void)
 {
- used_sems = 0;
- ipc_init_ids(&sem_ids,sc_semmni);
+ __sem_init_ns(&init_ipc_ns, &init_sem_ids);
  ipc_init_proc_interface("sysvipc/sem",
     "       key      semid perms      nsems   uid   gid  cuid  cgid      otime      ctime\n",
-    &sem_ids,
-    sysvipc_sem_proc_show);
+    IPC_SEM_IDS, sysvipc_sem_proc_show);
 }

 /*
@@ -163,7 +209,7 @@ void __init sem_init (void)
  */
 #define IN_WAKEUP 1

-static int newary (key_t key, int nsems, int semflg)
+static int newary (struct ipc_namespace *ns, key_t key, int nsems, int semflg)
 {
  int id;
  int retval;
@@ -172,7 +218,7 @@ static int newary (key_t key, int nsems,

  if (!nsems)
   return -EINVAL;
- if (used_sems + nsems > sc_semmns)
+ if (ns->used_sems + nsems > ns->sc_semmns)
```

```
   return -ENOSPC;

  size = sizeof (*sma) + nsems * sizeof (struct sem);
@@ -192,15 +238,15 @@ static int newary (key_t key, int nsems,
   return retval;
  }

- id = ipc_addid(&sem_ids, &sma->sem_perm, sc_semmni);
+ id = ipc_addid(&sem_ids(ns), &sma->sem_perm, ns->sc_semmni);
  if(id == -1) {
   security_sem_free(sma);
   ipc_rcu_putref(sma);
   return -ENOSPC;
  }
- used_sems += nsems;
+ ns->used_sems += nsems;

- sma->sem_id = sem_buildid(id, sma->sem_perm.seq);
+ sma->sem_id = sem_buildid(ns, id, sma->sem_perm.seq);
  sma->sem_base = (struct sem *) &sma[1];
  /* sma->sem_pending = NULL; */
  sma->sem_pending_last = &sma->sem_pending;
@@ -216,29 +262,32 @@ asmlinkage long sys_semget (key_t key, i
 {
  int id, err = -EINVAL;
  struct sem_array *sma;
+ struct ipc_namespace *ns;

- if (nsems < 0 || nsems > sc_semmsl)
+ ns = current->nsproxy->ipc_ns;
+
+ if (nsems < 0 || nsems > ns->sc_semmsl)
   return -EINVAL;
- mutex_lock(&sem_ids.mutex);
+ mutex_lock(&sem_ids(ns).mutex);

  if (key == IPC_PRIVATE) {
-  err = newary(key, nsems, semflg);
- } else if ((id = ipc_findkey(&sem_ids, key)) == -1) {  /* key not used */
+  err = newary(ns, key, nsems, semflg);
+ } else if ((id = ipc_findkey(&sem_ids(ns), key)) == -1) {  /* key not used */
   if (!(semflg & IPC_CREAT))
    err = -ENOENT;
   else
-   err = newary(key, nsems, semflg);
+   err = newary(ns, key, nsems, semflg);
  } else if (semflg & IPC_CREAT && semflg & IPC_EXCL) {
   err = -EEXIST;
```

```
  } else {
-  sma = sem_lock(id);
+  sma = sem_lock(ns, id);
    BUG_ON(sma==NULL);
    if (nsems > sma->sem_nsems)
     err = -EINVAL;
    else if (ipcperms(&sma->sem_perm, semflg))
     err = -EACCES;
    else {
-   int semid = sem_buildid(id, sma->sem_perm.seq);
+   int semid = sem_buildid(ns, id, sma->sem_perm.seq);
     err = security_sem_associate(sma, semflg);
     if (!err)
      err = semid;
@@ -246,7 +295,7 @@ asmlinkage long sys_semget (key_t key, i
    sem_unlock(sma);
  }

- mutex_unlock(&sem_ids.mutex);
+ mutex_unlock(&sem_ids(ns).mutex);
  return err;
 }

@@ -445,7 +494,7 @@ static int count_semzcnt (struct sem_arr
  * the spinlock for this semaphore set hold. sem_ids.mutex remains locked
  * on exit.
  */
-static void freeary (struct sem_array *sma, int id)
+static void freeary (struct ipc_namespace *ns, struct sem_array *sma, int id)
 {
  struct sem_undo *un;
  struct sem_queue *q;
@@ -473,10 +522,10 @@ static void freeary (struct sem_array *s
  }

  /* Remove the semaphore set from the ID array*/
- sma = sem_rmid(id);
+ sma = sem_rmid(ns, id);
  sem_unlock(sma);

- used_sems -= sma->sem_nsems;
+ ns->used_sems -= sma->sem_nsems;
  size = sizeof (*sma) + sma->sem_nsems * sizeof (struct sem);
  security_sem_free(sma);
  ipc_rcu_putref(sma);
@@ -504,7 +553,8 @@ static unsigned long copy_semid_to_user(
  }
 }
```

```
-static int semctl_nolock(int semid, int semnum, int cmd, int version, union semun arg)
+static int semctl_nolock(struct ipc_namespace *ns, int semid, int semnum,
+  int cmd, int version, union semun arg)
 {
  int err = -EINVAL;
  struct sem_array *sma;
@@ -521,24 +571,24 @@ static int semctl_nolock(int semid, int
    return err;

   memset(&seminfo,0,sizeof(seminfo));
-  seminfo.semmni = sc_semmni;
-  seminfo.semmns = sc_semmns;
-  seminfo.semmsl = sc_semmsl;
-  seminfo.semopm = sc_semopm;
+  seminfo.semmni = ns->sc_semmni;
+  seminfo.semmns = ns->sc_semmns;
+  seminfo.semmsl = ns->sc_semmsl;
+  seminfo.semopm = ns->sc_semopm;
   seminfo.semvmx = SEMVMX;
   seminfo.semmnu = SEMMNU;
   seminfo.semmap = SEMMAP;
   seminfo.semume = SEMUME;
-  mutex_lock(&sem_ids.mutex);
+  mutex_lock(&sem_ids(ns).mutex);
   if (cmd == SEM_INFO) {
-   seminfo.semusz = sem_ids.in_use;
-   seminfo.semaem = used_sems;
+   seminfo.semusz = sem_ids(ns).in_use;
+   seminfo.semaem = ns->used_sems;
   } else {
    seminfo.semusz = SEMUSZ;
    seminfo.semaem = SEMAEM;
   }
-  max_id = sem_ids.max_id;
-  mutex_unlock(&sem_ids.mutex);
+  max_id = sem_ids(ns).max_id;
+  mutex_unlock(&sem_ids(ns).mutex);
   if (copy_to_user (arg.__buf, &seminfo, sizeof(struct seminfo)))
    return -EFAULT;
   return (max_id < 0) ? 0: max_id;
@@ -548,12 +598,12 @@ static int semctl_nolock(int semid, int
   struct semid64_ds tbuf;
   int id;

-  if(semid >= sem_ids.entries->size)
+  if(semid >= sem_ids(ns).entries->size)
    return -EINVAL;
```

```
   memset(&tbuf,0,sizeof(tbuf));

-  sma = sem_lock(semid);
+  sma = sem_lock(ns, semid);
   if(sma == NULL)
    return -EINVAL;

@@ -565,7 +615,7 @@ static int semctl_nolock(int semid, int
   if (err)
    goto out_unlock;

-  id = sem_buildid(semid, sma->sem_perm.seq);
+  id = sem_buildid(ns, semid, sma->sem_perm.seq);

   kernel_to_ipc64_perm(&sma->sem_perm, &tbuf.sem_perm);
   tbuf.sem_otime  = sma->sem_otime;
@@ -585,7 +635,8 @@ out_unlock:
  return err;
 }

-static int semctl_main(int semid, int semnum, int cmd, int version, union semun arg)
+static int semctl_main(struct ipc_namespace *ns, int semid, int semnum,
+  int cmd, int version, union semun arg)
 {
  struct sem_array *sma;
  struct sem* curr;
@@ -594,14 +645,14 @@ static int semctl_main(int semid, int se
  ushort* sem_io = fast_sem_io;
  int nsems;

- sma = sem_lock(semid);
+ sma = sem_lock(ns, semid);
  if(sma==NULL)
   return -EINVAL;

  nsems = sma->sem_nsems;

  err=-EIDRM;
- if (sem_checkid(sma,semid))
+ if (sem_checkid(ns,sma,semid))
   goto out_unlock;

  err = -EACCES;
@@ -803,7 +854,8 @@ static inline unsigned long copy_semid_f
  }
 }
```

```
-static int semctl_down(int semid, int semnum, int cmd, int version, union semun arg)
+static int semctl_down(struct ipc_namespace *ns, int semid, int semnum,
+  int cmd, int version, union semun arg)
 {
  struct sem_array *sma;
  int err;
@@ -814,11 +866,11 @@ static int semctl_down(int semid, int se
   if(copy_semid_from_user (&setbuf, arg.buf, version))
    return -EFAULT;
  }
- sma = sem_lock(semid);
+ sma = sem_lock(ns, semid);
  if(sma==NULL)
   return -EINVAL;

- if (sem_checkid(sma,semid)) {
+ if (sem_checkid(ns,sma,semid)) {
   err=-EIDRM;
   goto out_unlock;
  }
@@ -845,7 +897,7 @@ static int semctl_down(int semid, int se

  switch(cmd){
  case IPC_RMID:
-  freeary(sma, semid);
+  freeary(ns, sma, semid);
   err = 0;
   break;
  case IPC_SET:
@@ -873,17 +925,19 @@ asmlinkage long sys_semctl (int semid, i
 {
  int err = -EINVAL;
  int version;
+ struct ipc_namespace *ns;

  if (semid < 0)
   return -EINVAL;

  version = ipc_parse_version(&cmd);
+ ns = current->nsproxy->ipc_ns;

  switch(cmd) {
  case IPC_INFO:
  case SEM_INFO:
  case SEM_STAT:
-  err = semctl_nolock(semid,semnum,cmd,version,arg);
+  err = semctl_nolock(ns,semid,semnum,cmd,version,arg);
   return err;
```

```
  case GETALL:
  case GETVAL:
@@ -893,13 +947,13 @@ asmlinkage long sys_semctl (int semid, i
  case IPC_STAT:
  case SETVAL:
  case SETALL:
-  err = semctl_main(semid,semnum,cmd,version,arg);
+  err = semctl_main(ns,semid,semnum,cmd,version,arg);
   return err;
  case IPC_RMID:
  case IPC_SET:
-  mutex_lock(&sem_ids.mutex);
-  err = semctl_down(semid,semnum,cmd,version,arg);
-  mutex_unlock(&sem_ids.mutex);
+  mutex_lock(&sem_ids(ns).mutex);
+  err = semctl_down(ns,semid,semnum,cmd,version,arg);
+  mutex_unlock(&sem_ids(ns).mutex);
   return err;
  default:
   return -EINVAL;
@@ -987,7 +1041,7 @@ static struct sem_undo *lookup_undo(stru
  return un;
 }

-static struct sem_undo *find_undo(int semid)
+static struct sem_undo *find_undo(struct ipc_namespace *ns, int semid)
 {
  struct sem_array *sma;
  struct sem_undo_list *ulp;
@@ -1006,12 +1060,12 @@ static struct sem_undo *find_undo(int se
   goto out;

  /* no undo structure around - allocate one. */
- sma = sem_lock(semid);
+ sma = sem_lock(ns, semid);
  un = ERR_PTR(-EINVAL);
  if(sma==NULL)
   goto out;
  un = ERR_PTR(-EIDRM);
- if (sem_checkid(sma,semid)) {
+ if (sem_checkid(ns,sma,semid)) {
   sem_unlock(sma);
   goto out;
  }
@@ -1071,10 +1125,13 @@ asmlinkage long sys_semtimedop(int semid
  int undos = 0, alter = 0, max;
  struct sem_queue queue;
  unsigned long jiffies_left = 0;
```

```
+ struct ipc_namespace *ns;
+
+ ns = current->nsproxy->ipc_ns;

  if (nsops < 1 || semid < 0)
   return -EINVAL;
- if (nsops > sc_semopm)
+ if (nsops > ns->sc_semopm)
   return -E2BIG;
  if(nsops > SEMOPM_FAST) {
   sops = kmalloc(sizeof(*sops)*nsops,GFP_KERNEL);
@@ -1110,7 +1167,7 @@ asmlinkage long sys_semtimedop(int semid

 retry_undos:
  if (undos) {
-  un = find_undo(semid);
+  un = find_undo(ns, semid);
   if (IS_ERR(un)) {
    error = PTR_ERR(un);
    goto out_free;
@@ -1118,12 +1175,12 @@ retry_undos:
  } else
   un = NULL;

- sma = sem_lock(semid);
+ sma = sem_lock(ns, semid);
  error=-EINVAL;
  if(sma==NULL)
   goto out_free;
  error = -EIDRM;
- if (sem_checkid(sma,semid))
+ if (sem_checkid(ns,sma,semid))
   goto out_unlock_free;
  /*
   * semid identifies are not unique - find_undo may have
@@ -1191,7 +1248,7 @@ retry_undos:
   goto out_free;
  }

- sma = sem_lock(semid);
+ sma = sem_lock(ns, semid);
  if(sma==NULL) {
   BUG_ON(queue.prev != NULL);
   error = -EIDRM;
@@ -1268,6 +1325,7 @@ void exit_sem(struct task_struct *tsk)
 {
  struct sem_undo_list *undo_list;
  struct sem_undo *u, **up;
```

```
+ struct ipc_namespace *ns;

  undo_list = tsk->sysvsem.undo_list;
  if (!undo_list)
@@ -1276,6 +1334,7 @@ void exit_sem(struct task_struct *tsk)
  if (!atomic_dec_and_test(&undo_list->refcnt))
   return;

+ ns = tsk->nsproxy->ipc_ns;
  /* There's no need to hold the semundo list lock, as current
         * is the last task exiting for this undo list.
   */
@@ -1289,14 +1348,14 @@ void exit_sem(struct task_struct *tsk)

   if(semid == -1)
    continue;
-  sma = sem_lock(semid);
+  sma = sem_lock(ns, semid);
   if (sma == NULL)
    continue;

   if (u->semid == -1)
    goto next_entry;

-  BUG_ON(sem_checkid(sma,u->semid));
+  BUG_ON(sem_checkid(ns,sma,u->semid));

   /* remove u from the sma->undo list */
   for (unp = &sma->undo; (un = *unp); unp = &un->id_next) {
```

Subject: [PATCH 5/6] IPC namespace - shm
Posted by Kirill Korotaev on Fri, 09 Jun 2006 15:09:56 GMT
View Forum Message <> Reply to Message

IPC namespace support for IPC shm code.

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

```
--- ./ipc/shm.c.ipcns 2006-06-06 14:47:58.000000000 +0400
+++ ./ipc/shm.c 2006-06-09 14:25:34.000000000 +0400
@@ -15,6 +15,10 @@
  *
  * support for audit of ipc object properties and permission changes
  * Dustin Kirkland <dustin.kirkland@us.ibm.com>
+ *
+ * namespaces support
```

```
+ * OpenVZ, SWsoft Inc.
+ * Pavel Emelianov <xemul@openvz.org>
  */

 #include <linux/config.h>
@@ -33,6 +37,7 @@
 #include <linux/ptrace.h>
 #include <linux/seq_file.h>
 #include <linux/mutex.h>
+#include <linux/nsproxy.h>

 #include <asm/uaccess.h>

@@ -41,59 +46,115 @@
 static struct file_operations shm_file_operations;
 static struct vm_operations_struct shm_vm_ops;

-static struct ipc_ids shm_ids;
+static struct ipc_ids init_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))
-#define shm_buildid(id, seq) \
- ipc_buildid(&shm_ids, id, seq)
+#define shm_ids(ns) (*((ns)->ids[IPC_SHM_IDS]))

-static int newseg (key_t key, int shmflg, size_t size);
+#define shm_lock(ns, id)  \
+ ((struct shmid_kernel*)ipc_lock(&shm_ids(ns),id))
+#define shm_unlock(shp)   \
+ ipc_unlock(&(shp)->shm_perm)
+#define shm_get(ns, id)   \
+ ((struct shmid_kernel*)ipc_get(&shm_ids(ns),id))
+#define shm_buildid(ns, id, seq) \
+ ipc_buildid(&shm_ids(ns), id, seq)
+
+static int newseg (struct ipc_namespace *ns, key_t key,
+  int shmflg, size_t size);
 static void shm_open (struct vm_area_struct *shmd);
 static void shm_close (struct vm_area_struct *shmd);
+static void shm_destroy (struct ipc_namespace *ns, struct shmid_kernel *shp);
 #ifdef CONFIG_PROC_FS
 static int sysvipc_shm_proc_show(struct seq_file *s, void *it);
 #endif

-size_t shm_ctlmax = SHMMAX;
-size_t  shm_ctlall = SHMALL;
```

```
-int  shm_ctlmni = SHMMNI;
+static void __ipc_init __shm_init_ns(struct ipc_namespace *ns, struct ipc_ids *ids)
+{
+ ns->ids[IPC_SHM_IDS] = ids;
+ ns->shm_ctlmax = SHMMAX;
+ ns->shm_ctlall = SHMALL;
+ ns->shm_ctlmni = SHMMNI;
+ ns->shm_tot = 0;
+ ipc_init_ids(ids, 1);
+}

-static int shm_tot; /* total number of shared memory pages */
+static void do_shm_rmid(struct ipc_namespace *ns, struct shmid_kernel *shp)
+{
+ if (shp->shm_nattch){
+  shp->shm_perm.mode |= SHM_DEST;
+  /* Do not find it any more */
+  shp->shm_perm.key = IPC_PRIVATE;
+  shm_unlock(shp);
+ } else
+  shm_destroy(ns, shp);
+}
+
+#ifdef CONFIG_IPC_NS
+int shm_init_ns(struct ipc_namespace *ns)
+{
+ struct ipc_ids *ids;
+
+ ids = kmalloc(sizeof(struct ipc_ids), GFP_KERNEL);
+ if (ids == NULL)
+  return -ENOMEM;
+
+ __shm_init_ns(ns, ids);
+ return 0;
+}
+
+void shm_exit_ns(struct ipc_namespace *ns)
+{
+ int i;
+ struct shmid_kernel *shp;
+
+ mutex_lock(&shm_ids(ns).mutex);
+ for (i = 0; i <= shm_ids(ns).max_id; i++) {
+  shp = shm_lock(ns, i);
+  if (shp == NULL)
+   continue;
+
+  do_shm_rmid(ns, shp);
```

```
+ }
+ mutex_unlock(&shm_ids(ns).mutex);
+
+ kfree(ns->ids[IPC_SHM_IDS]);
+ ns->ids[IPC_SHM_IDS] = NULL;
+}
+#endif

 void __init shm_init (void)
 {
- ipc_init_ids(&shm_ids, 1);
+ __shm_init_ns(&init_ipc_ns, &init_shm_ids);
  ipc_init_proc_interface("sysvipc/shm",
     "       key      shmid perms       size  cpid  lpid nattch   uid   gid  cuid  cgid      atime      dtime
ctime\n",
-    &shm_ids,
-    sysvipc_shm_proc_show);
+    IPC_SHM_IDS, sysvipc_shm_proc_show);
 }

-static inline int shm_checkid(struct shmid_kernel *s, int id)
+static inline int shm_checkid(struct ipc_namespace *ns,
+  struct shmid_kernel *s, int id)
 {
- if (ipc_checkid(&shm_ids,&s->shm_perm,id))
+ if (ipc_checkid(&shm_ids(ns), &s->shm_perm, id))
   return -EIDRM;
  return 0;
 }

-static inline struct shmid_kernel *shm_rmid(int id)
+static inline struct shmid_kernel *shm_rmid(struct ipc_namespace *ns, int id)
 {
- return (struct shmid_kernel *)ipc_rmid(&shm_ids,id);
+ return (struct shmid_kernel *)ipc_rmid(&shm_ids(ns), id);
 }

-static inline int shm_addid(struct shmid_kernel *shp)
+static inline int shm_addid(struct ipc_namespace *ns, struct shmid_kernel *shp)
 {
- return ipc_addid(&shm_ids, &shp->shm_perm, shm_ctlmni);
+ return ipc_addid(&shm_ids(ns), &shp->shm_perm, ns->shm_ctlmni);
 }



-static inline void shm_inc (int id) {
+static inline void shm_inc(struct ipc_namespace *ns, int id)
```

```
+{
  struct shmid_kernel *shp;

- shp = shm_lock(id);
+ shp = shm_lock(ns, id);
  BUG_ON(!shp);
  shp->shm_atim = get_seconds();
  shp->shm_lprid = current->tgid;
@@ -101,10 +162,13 @@ static inline void shm_inc (int id) {
  shm_unlock(shp);
 }

+#define shm_file_ns(file) (*((struct ipc_namespace **)&(file)->private_data))
+
 /* This is called by fork, once for every shm attach. */
-static void shm_open (struct vm_area_struct *shmd)
+static void shm_open(struct vm_area_struct *shmd)
 {
- shm_inc (shmd->vm_file->f_dentry->d_inode->i_ino);
+ shm_inc(shm_file_ns(shmd->vm_file),
+    shmd->vm_file->f_dentry->d_inode->i_ino);
 }

 /*
@@ -115,10 +179,10 @@ static void shm_open (struct vm_area_str
  * It has to be called with shp and shm_ids.mutex locked,
  * but returns with shp unlocked and freed.
  */
-static void shm_destroy (struct shmid_kernel *shp)
+static void shm_destroy(struct ipc_namespace *ns, struct shmid_kernel *shp)
 {
- shm_tot -= (shp->shm_segsz + PAGE_SIZE - 1) >> PAGE_SHIFT;
- shm_rmid (shp->id);
+ ns->shm_tot -= (shp->shm_segsz + PAGE_SIZE - 1) >> PAGE_SHIFT;
+ shm_rmid(ns, shp->id);
  shm_unlock(shp);
  if (!is_file_hugepages(shp->shm_file))
   shmem_lock(shp->shm_file, 0, shp->mlock_user);
@@ -141,20 +205,23 @@ static void shm_close (struct vm_area_st
  struct file * file = shmd->vm_file;
  int id = file->f_dentry->d_inode->i_ino;
  struct shmid_kernel *shp;
+ struct ipc_namespace *ns;

- mutex_lock(&shm_ids.mutex);
+ ns = shm_file_ns(file);
+
+ mutex_lock(&shm_ids(ns).mutex);
```

```
  /* remove from the list of attaches of the shm segment */
- shp = shm_lock(id);
+ shp = shm_lock(ns, id);
  BUG_ON(!shp);
  shp->shm_lprid = current->tgid;
  shp->shm_dtim = get_seconds();
  shp->shm_nattch--;
  if(shp->shm_nattch == 0 &&
     shp->shm_perm.mode & SHM_DEST)
-  shm_destroy (shp);
+  shm_destroy(ns, shp);
  else
   shm_unlock(shp);
- mutex_unlock(&shm_ids.mutex);
+ mutex_unlock(&shm_ids(ns).mutex);
 }

 static int shm_mmap(struct file * file, struct vm_area_struct * vma)
@@ -166,14 +233,25 @@ static int shm_mmap(struct file * file,
   vma->vm_ops = &shm_vm_ops;
   if (!(vma->vm_flags & VM_WRITE))
    vma->vm_flags &= ~VM_MAYWRITE;
-  shm_inc(file->f_dentry->d_inode->i_ino);
+  shm_inc(shm_file_ns(file), file->f_dentry->d_inode->i_ino);
  }

  return ret;
 }

+static int shm_release(struct inode *ino, struct file *file)
+{
+ struct ipc_namespace *ns;
+
+ ns = shm_file_ns(file);
+ put_ipc_ns(ns);
+ shm_file_ns(file) = NULL;
+ return 0;
+}
+
 static struct file_operations shm_file_operations = {
- .mmap = shm_mmap,
+ .mmap  = shm_mmap,
+ .release = shm_release,
 #ifndef CONFIG_MMU
  .get_unmapped_area = shmem_get_unmapped_area,
 #endif
@@ -189,7 +267,7 @@ static struct vm_operations_struct shm_v
 #endif
```

```
 };

-static int newseg (key_t key, int shmflg, size_t size)
+static int newseg (struct ipc_namespace *ns, key_t key, int shmflg, size_t size)
 {
  int error;
  struct shmid_kernel *shp;
@@ -198,10 +276,10 @@ static int newseg (key_t key, int shmflg
  char name[13];
  int id;

- if (size < SHMMIN || size > shm_ctlmax)
+ if (size < SHMMIN || size > ns->shm_ctlmax)
   return -EINVAL;

- if (shm_tot + numpages >= shm_ctlall)
+ if (ns->shm_tot + numpages >= ns->shm_ctlall)
   return -ENOSPC;

  shp = ipc_rcu_alloc(sizeof(*shp));
@@ -240,7 +318,7 @@ static int newseg (key_t key, int shmflg
   goto no_file;

  error = -ENOSPC;
- id = shm_addid(shp);
+ id = shm_addid(ns, shp);
  if(id == -1)
   goto no_id;

@@ -250,15 +328,17 @@ static int newseg (key_t key, int shmflg
  shp->shm_ctim = get_seconds();
  shp->shm_segsz = size;
  shp->shm_nattch = 0;
- shp->id = shm_buildid(id,shp->shm_perm.seq);
+ shp->id = shm_buildid(ns, id, shp->shm_perm.seq);
  shp->shm_file = file;
  file->f_dentry->d_inode->i_ino = shp->id;

+ shm_file_ns(file) = get_ipc_ns(ns);
+
  /* Hugetlb ops would have already been assigned. */
  if (!(shmflg & SHM_HUGETLB))
   file->f_op = &shm_file_operations;

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

```
@@ -274,33 +354,36 @@ asmlinkage long sys_shmget (key_t key, s
 {
  struct shmid_kernel *shp;
  int err, id = 0;
+ struct ipc_namespace *ns;

- mutex_lock(&shm_ids.mutex);
+ ns = current->nsproxy->ipc_ns;
+
+ mutex_lock(&shm_ids(ns).mutex);
  if (key == IPC_PRIVATE) {
-  err = newseg(key, shmflg, size);
- } else if ((id = ipc_findkey(&shm_ids, key)) == -1) {
+  err = newseg(ns, key, shmflg, size);
+ } else if ((id = ipc_findkey(&shm_ids(ns), key)) == -1) {
   if (!(shmflg & IPC_CREAT))
    err = -ENOENT;
   else
-   err = newseg(key, shmflg, size);
+   err = newseg(ns, key, shmflg, size);
  } else if ((shmflg & IPC_CREAT) && (shmflg & IPC_EXCL)) {
   err = -EEXIST;
  } else {
-  shp = shm_lock(id);
+  shp = shm_lock(ns, id);
   BUG_ON(shp==NULL);
   if (shp->shm_segsz < size)
    err = -EINVAL;
   else if (ipcperms(&shp->shm_perm, shmflg))
    err = -EACCES;
   else {
-   int shmid = shm_buildid(id, shp->shm_perm.seq);
+   int shmid = shm_buildid(ns, id, shp->shm_perm.seq);
    err = security_shm_associate(shp, shmflg);
    if (!err)
     err = shmid;
   }
   shm_unlock(shp);
  }
- mutex_unlock(&shm_ids.mutex);
+ mutex_unlock(&shm_ids(ns).mutex);

  return err;
 }
@@ -396,18 +479,19 @@ static inline unsigned long copy_shminfo
  }
 }
```

```
-static void shm_get_stat(unsigned long *rss, unsigned long *swp)
+static void shm_get_stat(struct ipc_namespace *ns, unsigned long *rss,
+  unsigned long *swp)
 {
  int i;

  *rss = 0;
  *swp = 0;

- for (i = 0; i <= shm_ids.max_id; i++) {
+ for (i = 0; i <= shm_ids(ns).max_id; i++) {
   struct shmid_kernel *shp;
   struct inode *inode;

-  shp = shm_get(i);
+  shp = shm_get(ns, i);
   if(!shp)
    continue;

@@ -431,6 +515,7 @@ asmlinkage long sys_shmctl (int shmid, i
  struct shm_setbuf setbuf;
  struct shmid_kernel *shp;
  int err, version;
+ struct ipc_namespace *ns;

  if (cmd < 0 || shmid < 0) {
   err = -EINVAL;
@@ -438,6 +523,7 @@ asmlinkage long sys_shmctl (int shmid, i
  }

  version = ipc_parse_version(&cmd);
+ ns = current->nsproxy->ipc_ns;

  switch (cmd) { /* replace with proc interface ? */
  case IPC_INFO:
@@ -449,15 +535,15 @@ asmlinkage long sys_shmctl (int shmid, i
    return err;

   memset(&shminfo,0,sizeof(shminfo));
-  shminfo.shmmni = shminfo.shmseg = shm_ctlmni;
-  shminfo.shmmax = shm_ctlmax;
-  shminfo.shmall = shm_ctlall;
+  shminfo.shmmni = shminfo.shmseg = ns->shm_ctlmni;
+  shminfo.shmmax = ns->shm_ctlmax;
+  shminfo.shmall = ns->shm_ctlall;

   shminfo.shmmin = SHMMIN;
```

```
   if(copy_shminfo_to_user (buf, &shminfo, version))
    return -EFAULT;
   /* reading a integer is always atomic */
-  err= shm_ids.max_id;
+  err= shm_ids(ns).max_id;
   if(err<0)
    err = 0;
   goto out;
@@ -471,14 +557,14 @@ asmlinkage long sys_shmctl (int shmid, i
    return err;

   memset(&shm_info,0,sizeof(shm_info));
-  mutex_lock(&shm_ids.mutex);
-  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;
+  mutex_lock(&shm_ids(ns).mutex);
+  shm_info.used_ids = shm_ids(ns).in_use;
+  shm_get_stat (ns, &shm_info.shm_rss, &shm_info.shm_swp);
+  shm_info.shm_tot = ns->shm_tot;
   shm_info.swap_attempts = 0;
   shm_info.swap_successes = 0;
-  err = shm_ids.max_id;
-  mutex_unlock(&shm_ids.mutex);
+  err = shm_ids(ns).max_id;
+  mutex_unlock(&shm_ids(ns).mutex);
   if(copy_to_user (buf, &shm_info, sizeof(shm_info))) {
    err = -EFAULT;
    goto out;
@@ -493,17 +579,17 @@ asmlinkage long sys_shmctl (int shmid, i
   struct shmid64_ds tbuf;
   int result;
   memset(&tbuf, 0, sizeof(tbuf));
-  shp = shm_lock(shmid);
+  shp = shm_lock(ns, shmid);
   if(shp==NULL) {
    err = -EINVAL;
    goto out;
   } else if(cmd==SHM_STAT) {
    err = -EINVAL;
-   if (shmid > shm_ids.max_id)
+   if (shmid > shm_ids(ns).max_id)
     goto out_unlock;
-   result = shm_buildid(shmid, shp->shm_perm.seq);
+   result = shm_buildid(ns, shmid, shp->shm_perm.seq);
   } else {
-   err = shm_checkid(shp,shmid);
+   err = shm_checkid(ns, shp,shmid);
```

```
    if(err)
     goto out_unlock;
    result = 0;
@@ -535,12 +621,12 @@ asmlinkage long sys_shmctl (int shmid, i
  case SHM_LOCK:
  case SHM_UNLOCK:
  {
-  shp = shm_lock(shmid);
+  shp = shm_lock(ns, shmid);
   if(shp==NULL) {
    err = -EINVAL;
    goto out;
   }
-  err = shm_checkid(shp,shmid);
+  err = shm_checkid(ns, shp,shmid);
   if(err)
    goto out_unlock;

@@ -591,12 +677,12 @@ asmlinkage long sys_shmctl (int shmid, i
    * Instead we set a destroyed flag, and then blow
    * the name away when the usage hits zero.
    */
-  mutex_lock(&shm_ids.mutex);
-  shp = shm_lock(shmid);
+  mutex_lock(&shm_ids(ns).mutex);
+  shp = shm_lock(ns, shmid);
   err = -EINVAL;
   if (shp == NULL)
    goto out_up;
-  err = shm_checkid(shp, shmid);
+  err = shm_checkid(ns, shp, shmid);
   if(err)
    goto out_unlock_up;

@@ -615,14 +701,8 @@ asmlinkage long sys_shmctl (int shmid, i
   if (err)
    goto out_unlock_up;

-  if (shp->shm_nattch){
-   shp->shm_perm.mode |= SHM_DEST;
-   /* Do not find it any more */
-   shp->shm_perm.key = IPC_PRIVATE;
-   shm_unlock(shp);
-  } else
-   shm_destroy (shp);
-  mutex_unlock(&shm_ids.mutex);
+  do_shm_rmid(ns, shp);
+  mutex_unlock(&shm_ids(ns).mutex);
```

```
   goto out;
  }

@@ -632,12 +712,12 @@ asmlinkage long sys_shmctl (int shmid, i
    err = -EFAULT;
    goto out;
   }
-  mutex_lock(&shm_ids.mutex);
-  shp = shm_lock(shmid);
+  mutex_lock(&shm_ids(ns).mutex);
+  shp = shm_lock(ns, shmid);
   err=-EINVAL;
   if(shp==NULL)
    goto out_up;
-  err = shm_checkid(shp,shmid);
+  err = shm_checkid(ns, shp,shmid);
   if(err)
    goto out_unlock_up;
   err = audit_ipc_obj(&(shp->shm_perm));
@@ -674,7 +754,7 @@ asmlinkage long sys_shmctl (int shmid, i
 out_unlock_up:
  shm_unlock(shp);
 out_up:
- mutex_unlock(&shm_ids.mutex);
+ mutex_unlock(&shm_ids(ns).mutex);
  goto out;
 out_unlock:
  shm_unlock(shp);
@@ -700,6 +780,7 @@ long do_shmat(int shmid, char __user *sh
  unsigned long prot;
  int acc_mode;
  void *user_addr;
+ struct ipc_namespace *ns;

  if (shmid < 0) {
   err = -EINVAL;
@@ -738,12 +819,13 @@ long do_shmat(int shmid, char __user *sh
   * We cannot rely on the fs check since SYSV IPC does have an
   * additional creator id...
   */
- shp = shm_lock(shmid);
+ ns = current->nsproxy->ipc_ns;
+ shp = shm_lock(ns, shmid);
  if(shp == NULL) {
   err = -EINVAL;
   goto out;
  }
- err = shm_checkid(shp,shmid);
```

```
+ err = shm_checkid(ns, shp,shmid);
  if (err) {
   shm_unlock(shp);
   goto out;
@@ -784,16 +866,16 @@ long do_shmat(int shmid, char __user *sh
 invalid:
  up_write(&current->mm->mmap_sem);

- mutex_lock(&shm_ids.mutex);
- shp = shm_lock(shmid);
+ mutex_lock(&shm_ids(ns).mutex);
+ shp = shm_lock(ns, shmid);
  BUG_ON(!shp);
  shp->shm_nattch--;
  if(shp->shm_nattch == 0 &&
     shp->shm_perm.mode & SHM_DEST)
-  shm_destroy (shp);
+  shm_destroy(ns, shp);
  else
   shm_unlock(shp);
- mutex_unlock(&shm_ids.mutex);
+ mutex_unlock(&shm_ids(ns).mutex);

  *raddr = (unsigned long) user_addr;
  err = 0;
```

Subject: [PATCH 6/6] IPC namespace - sysctls
Posted by Kirill Korotaev on Fri, 09 Jun 2006 15:11:07 GMT
View Forum Message <> Reply to Message

Sysctl tweaks for IPC namespace

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

```
--- ./kernel/sysctl.c.ipcns 2006-06-06 14:47:59.000000000 +0400
+++ ./kernel/sysctl.c 2006-06-08 15:29:58.000000000 +0400
@@ -104,13 +104,8 @@ extern char modprobe_path[];
 extern int sg_big_buff;
 #endif
 #ifdef CONFIG_SYSVIPC
-extern size_t shm_ctlmax;
-extern size_t shm_ctlall;
-extern int shm_ctlmni;
-extern int msg_ctlmax;
-extern int msg_ctlmnb;
-extern int msg_ctlmni;
```

```
-extern int sem_ctls[];
+static int proc_do_ipc_string(ctl_table *table, int write, struct file *filp,
+  void __user *buffer, size_t *lenp, loff_t *ppos);
 #endif

 #ifdef __sparc__
@@ -510,58 +505,58 @@ static ctl_table kern_table[] = {
  {
   .ctl_name = KERN_SHMMAX,
   .procname = "shmmax",
-  .data  = &shm_ctlmax,
+  .data  = NULL,
   .maxlen  = sizeof (size_t),
   .mode  = 0644,
-  .proc_handler = &proc_doulongvec_minmax,
+  .proc_handler = &proc_do_ipc_string,
  },
  {
   .ctl_name = KERN_SHMALL,
   .procname = "shmall",
-  .data  = &shm_ctlall,
+  .data  = NULL,
   .maxlen  = sizeof (size_t),
   .mode  = 0644,
-  .proc_handler = &proc_doulongvec_minmax,
+  .proc_handler = &proc_do_ipc_string,
  },
  {
   .ctl_name = KERN_SHMMNI,
   .procname = "shmmni",
-  .data  = &shm_ctlmni,
+  .data  = NULL,
   .maxlen  = sizeof (int),
   .mode  = 0644,
-  .proc_handler = &proc_dointvec,
+  .proc_handler = &proc_do_ipc_string,
  },
  {
   .ctl_name = KERN_MSGMAX,
   .procname = "msgmax",
-  .data  = &msg_ctlmax,
+  .data  = NULL,
   .maxlen  = sizeof (int),
   .mode  = 0644,
-  .proc_handler = &proc_dointvec,
+  .proc_handler = &proc_do_ipc_string,
  },
  {
```

```
   .ctl_name = KERN_MSGMNI,
   .procname = "msgmni",
-  .data  = &msg_ctlmni,
+  .data  = NULL,
   .maxlen  = sizeof (int),
   .mode  = 0644,
-  .proc_handler = &proc_dointvec,
+  .proc_handler = &proc_do_ipc_string,
  },
  {
   .ctl_name = KERN_MSGMNB,
   .procname =  "msgmnb",
-  .data  = &msg_ctlmnb,
+  .data  = NULL,
   .maxlen  = sizeof (int),
   .mode  = 0644,
-  .proc_handler = &proc_dointvec,
+  .proc_handler = &proc_do_ipc_string,
  },
  {
   .ctl_name = KERN_SEM,
   .procname = "sem",
-  .data  = &sem_ctls,
+  .data  = NULL,
   .maxlen  = 4*sizeof (int),
   .mode  = 0644,
-  .proc_handler = &proc_dointvec,
+  .proc_handler = &proc_do_ipc_string,
  },
 #endif
 #ifdef CONFIG_MAGIC_SYSRQ
@@ -1866,8 +1861,9 @@ static int do_proc_dointvec_conv(int *ne
  return 0;
 }

-static int do_proc_dointvec(ctl_table *table, int write, struct file *filp,
-    void __user *buffer, size_t *lenp, loff_t *ppos,
+static int __do_proc_dointvec(void *tbl_data, ctl_table *table,
+    int write, struct file *filp, void __user *buffer,
+    size_t *lenp, loff_t *ppos,
     int (*conv)(int *negp, unsigned long *lvalp, int *valp,
         int write, void *data),
     void *data)
@@ -1880,13 +1876,13 @@ static int do_proc_dointvec(ctl_table *t
  char buf[TMPBUFLEN], *p;
  char __user *s = buffer;

- if (!table->data || !table->maxlen || !*lenp ||
```

```
+ if (!tbl_data || !table->maxlen || !*lenp ||
      (*ppos && !write)) {
   *lenp = 0;
   return 0;
  }

- i = (int *) table->data;
+ i = (int *) tbl_data;
  vleft = table->maxlen / sizeof(*i);
  left = *lenp;

@@ -1975,6 +1971,16 @@ static int do_proc_dointvec(ctl_table *t
 #undef TMPBUFLEN
 }

+static int do_proc_dointvec(ctl_table *table, int write, struct file *filp,
+    void __user *buffer, size_t *lenp, loff_t *ppos,
+    int (*conv)(int *negp, unsigned long *lvalp, int *valp,
+         int write, void *data),
+    void *data)
+{
+ return __do_proc_dointvec(table->data, table, write, filp,
+   buffer, lenp, ppos, conv, data);
+}
+
 /**
  * proc_dointvec - read a vector of integers
  * @table: the sysctl table
@@ -2108,7 +2114,7 @@ int proc_dointvec_minmax(ctl_table *tabl
     do_proc_dointvec_minmax_conv, &param);
 }

-static int do_proc_doulongvec_minmax(ctl_table *table, int write,
+static int __do_proc_doulongvec_minmax(void *data, ctl_table *table, int write,
         struct file *filp,
         void __user *buffer,
         size_t *lenp, loff_t *ppos,
@@ -2122,13 +2128,13 @@ static int do_proc_doulongvec_minmax(ctl
  char buf[TMPBUFLEN], *p;
  char __user *s = buffer;

- if (!table->data || !table->maxlen || !*lenp ||
+ if (!data || !table->maxlen || !*lenp ||
      (*ppos && !write)) {
   *lenp = 0;
   return 0;
  }
```

```
- i = (unsigned long *) table->data;
+ i = (unsigned long *) data;
  min = (unsigned long *) table->extra1;
  max = (unsigned long *) table->extra2;
  vleft = table->maxlen / sizeof(unsigned long);
@@ -2213,6 +2219,17 @@ static int do_proc_doulongvec_minmax(ctl
 #undef TMPBUFLEN
 }

+static int do_proc_doulongvec_minmax(ctl_table *table, int write,
+         struct file *filp,
+         void __user *buffer,
+         size_t *lenp, loff_t *ppos,
+         unsigned long convmul,
+         unsigned long convdiv)
+{
+ return __do_proc_doulongvec_minmax(table->data, table, write,
+    filp, buffer, lenp, ppos, convmul, convdiv);
+}
+
 /**
  * proc_doulongvec_minmax - read a vector of long integers with min/max values
  * @table: the sysctl table
@@ -2401,6 +2418,49 @@ int proc_dointvec_ms_jiffies(ctl_table *
     do_proc_dointvec_ms_jiffies_conv, NULL);
 }

+#ifdef CONFIG_SYSVIPC
+static int proc_do_ipc_string(ctl_table *table, int write, struct file *filp,
+  void __user *buffer, size_t *lenp, loff_t *ppos)
+{
+ void *data;
+ struct ipc_namespace *ns;
+
+ ns = current->nsproxy->ipc_ns;
+
+ switch (table->ctl_name) {
+ case KERN_SHMMAX:
+  data = &ns->shm_ctlmax;
+  goto proc_minmax;
+ case KERN_SHMALL:
+  data = &ns->shm_ctlall;
+  goto proc_minmax;
+ case KERN_SHMMNI:
+  data = &ns->shm_ctlmni;
+  break;
+ case KERN_MSGMAX:
+  data = &ns->msg_ctlmax;
```

```
+  break;
+ case KERN_MSGMNI:
+  data = &ns->msg_ctlmni;
+  break;
+ case KERN_MSGMNB:
+  data = &ns->msg_ctlmnb;
+  break;
+ case KERN_SEM:
+  data = &ns->sem_ctls;
+  break;
+ default:
+  return -EINVAL;
+ }
+
+ return __do_proc_dointvec(data, table, write, filp, buffer,
+   lenp, ppos, NULL, NULL);
+proc_minmax:
+ return __do_proc_doulongvec_minmax(data, table, write, filp, buffer,
+   lenp, ppos, 1l, 1l);
+}
+#endif
+
 #else /* CONFIG_PROC_FS */

 int proc_dostring(ctl_table *table, int write, struct file *filp,
```

Subject: Re: [PATCH 1/6] IPC namespace core
Posted by Cedric Le Goater on Fri, 09 Jun 2006 15:20:15 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
> This patch implements core IPC namespace changes:
> - ipc_namespace structure
> - new config option CONFIG_IPC_NS
> - adds CLONE_NEWIPC flag
> - unshare support
>
> Signed-Off-By: Pavel Emelianov <xemul@openvz.org>
> Signed-Off-By: Kirill Korotaev <dev@openvz.org>
>
>
> ---------- ----------
>
> --- ./include/linux/init_task.h.ipcns 2006-06-06 14:47:58.000000000 +0400
> +++ ./include/linux/init_task.h 2006-06-08 14:28:23.000000000 +0400
> @@ -73,6 +73,7 @@ extern struct nsproxy init_nsproxy;
>   .count = ATOMIC_INIT(1),   \

```
>   .nslock  = SPIN_LOCK_UNLOCKED,    \
>   .uts_ns  = &init_uts_ns,     \
> + .ipc_ns  = &init_ipc_ns,     \
>   .namespace = NULL,      \
>  }
>
> --- ./include/linux/ipc.h.ipcns 2006-04-21 11:59:36.000000000 +0400
> +++ ./include/linux/ipc.h 2006-06-08 15:43:43.000000000 +0400
> @@ -2,6 +2,7 @@
>  #define _LINUX_IPC_H
>
>  #include <linux/types.h>
> +#include <linux/kref.h>
>
>  #define IPC_PRIVATE ((__kernel_key_t) 0)
>
> @@ -68,6 +69,41 @@ struct kern_ipc_perm
>   void  *security;
>  };
>
> +struct ipc_ids;
> +struct ipc_namespace {
> + struct kref kref;
> + struct ipc_ids *ids[3];
> +
> + int  sem_ctls[4];
> + int  used_sems;
> +
> + int  msg_ctlmax;
> + int  msg_ctlmnb;
> + int  msg_ctlmni;
> +
> + size_t  shm_ctlmax;
> + size_t  shm_ctlall;
> + int  shm_ctlmni;
> + int  shm_tot;
> +};
```

you could probably simplify your patch by moving struct ipc_ids to ipc.h
and not allocating ids.

see patch bellow. I've been working all week on this patchset :)

C.

Subject: Re: [PATCH 1/6] IPC namespace core

Posted by James Morris on Fri, 09 Jun 2006 15:26:46 GMT
View Forum Message <> Reply to Message

On Fri, 9 Jun 2006, Kirill Korotaev wrote:

> This patch implements core IPC namespace changes:
> - ipc_namespace structure
> - new config option CONFIG_IPC_NS
> - adds CLONE_NEWIPC flag
> - unshare support
>

Please post patches as inline text, so they can be reviewed inline.

+struct ipc_namespace {
+	struct kref	kref;
+	struct ipc_ids	*ids[3];
+
+	int		sem_ctls[4];

It'd be nice if these weren't magic numbers.

--
James Morris
<jmorris@namei.org>

Subject: Re: [PATCH 1/6] IPC namespace core
Posted by Andrew Morton on Fri, 09 Jun 2006 18:38:02 GMT
View Forum Message <> Reply to Message

On Fri, 09 Jun 2006 19:01:38 +0400
Kirill Korotaev <dev@openvz.org> wrote:

> --- ./include/linux/init_task.h.ipcns 2006-06-06 14:47:58.000000000 +0400
> +++ ./include/linux/init_task.h 2006-06-08 14:28:23.000000000 +0400
> @@ -73,6 +73,7 @@ extern struct nsproxy init_nsproxy;
> .count = ATOMIC_INIT(1), \
> .nslock = SPIN_LOCK_UNLOCKED, \
> .uts_ns = &init_uts_ns, \
> + .ipc_ns = &init_ipc_ns, \
> .namespace = NULL, \
> }

Not all the new additions here are inside CONFIG_IPC_NS. It'd be nice to
make it so.

Subject: Re: [PATCH 1/6] IPC namespace core
Posted by ebiederm on Sat, 10 Jun 2006 00:44:30 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@openvz.org> writes:

> --- ./kernel/fork.c.ipcns 2006-06-06 14:47:58.000000000 +0400
> +++ ./kernel/fork.c 2006-06-08 15:31:03.000000000 +0400
> @@ -1592,6 +1592,7 @@ asmlinkage long sys_unshare(unsigned lon
> struct sem_undo_list *new_ulist = NULL;
> struct nsproxy *new_nsproxy = NULL, *old_nsproxy = NULL;
> struct uts_namespace *uts, *new_uts = NULL;
> + struct ipc_namespace *ipc, *new_ipc = NULL;
>
> check_unshare_flags(&unshare_flags);
>
> @@ -1617,18 +1618,20 @@ asmlinkage long sys_unshare(unsigned lon
> goto bad_unshare_cleanup_fd;
> if ((err = unshare_utsname(unshare_flags, &new_uts)))
> goto bad_unshare_cleanup_semundo;
> + if ((err = unshare_ipcs(unshare_flags, &new_ipc)))
> + goto bad_unshare_cleanup_uts;
>
> if (new_ns || new_uts) {
This test needs to be updated to test for new_ipc.
> old_nsproxy = current->nsproxy;
> new_nsproxy = dup_namespaces(old_nsproxy);
> if (!new_nsproxy) {
> err = -ENOMEM;
> - goto bad_unshare_cleanup_uts;
> + goto bad_unshare_cleanup_ipc;
> }
> }

Eric

Subject: Re: [PATCH 1/6] IPC namespace core
Posted by Andrew Morton on Sat, 10 Jun 2006 03:22:58 GMT
View Forum Message <> Reply to Message

On Fri, 09 Jun 2006 19:01:38 +0400
Kirill Korotaev <dev@openvz.org> wrote:

> This patch implements core IPC namespace changes:
> - ipc_namespace structure
> - new config option CONFIG_IPC_NS
> - adds CLONE_NEWIPC flag

> - unshare support

`make allnoconfig'

arch/i386/kernel/init_task.o:(.data+0x1a0): undefined reference to `init_ipc_ns'
kernel/built-in.o: In function `free_nsproxy':
: undefined reference to `free_ipc_ns'
kernel/built-in.o: In function `copy_namespaces':
: undefined reference to `copy_ipcs'

Subject: Re: [PATCH 2/6] IPC namespace - utils
Posted by Cedric Le Goater on Mon, 12 Jun 2006 17:08:38 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:

> +static struct ipc_namespace *clone_ipc_ns(struct ipc_namespace *old_ns)
> +{
> + int err;
> + struct ipc_namespace *ns;
> +
> + err = -ENOMEM;
> + ns = kmalloc(sizeof(struct ipc_namespace), GFP_KERNEL);
> + if (ns == NULL)
> + goto err_mem;
> +
> + err = sem_init_ns(ns);
> + if (err)
> + goto err_sem;
> + err = msg_init_ns(ns);
> + if (err)
> + goto err_msg;
> + err = shm_init_ns(ns);
> + if (err)
> + goto err_shm;
> +
> + kref_init(&ns->kref);
> + return ns;
> +
> +err_shm:
> + msg_exit_ns(ns);
> +err_msg:
> + sem_exit_ns(ns);
> +err_sem:
> + kfree(ns);
> +err_mem:
> + return ERR_PTR(err);

> +}

I've used the ipc namespace patchset in rc6-mm2. Thanks for putting this
together, it works pretty well ! A few questions when we clone :

* We should do something close to what exit_sem() already does to clear the
sem_undo list from the task doing the clone() or unshare().

* I don't like the idea of being able to unshare the ipc namespace and keep
some shared memory from the previous ipc namespace mapped in the process mm.
Should we forbid the unshare ?

Small fix follows,

thanks,

C.

Subject: Re: [PATCH] IPC namespace
Posted by Dave Hansen on Mon, 12 Jun 2006 17:19:28 GMT
View Forum Message <> Reply to Message

On Fri, 2006-06-09 at 18:55 +0400, Kirill Korotaev wrote:
> The patches in this thread add IPC namespace functionality
> additionally to already included in -mm tree UTS namespace.
>
> This patch set allows to unshare IPCs and have a private set
> of IPC objects (sem, shm, msg) inside namespace. Basically, it is
> another building block of containers functionality.
>
> Tested with LTP inside namespaces.

Do you, by chance, have any test cases for this code that test the
unsharing itself, and not just the functionality before and after an
unshare?

-- Dave

Subject: Re: [PATCH 2/6] IPC namespace - utils
Posted by ebiederm on Mon, 12 Jun 2006 18:01:48 GMT
View Forum Message <> Reply to Message

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

> I've used the ipc namespace patchset in rc6-mm2. Thanks for putting this

> together, it works pretty well ! A few questions when we clone :
>
> * We should do something close to what exit_sem() already does to clear the
> sem_undo list from the task doing the clone() or unshare().

Possibly which case are you trying to prevent?

> * I don't like the idea of being able to unshare the ipc namespace and keep
> some shared memory from the previous ipc namespace mapped in the process mm.
> Should we forbid the unshare ?

No. As long as the code handles that case properly we should be fine.
As a general principle we should be able to keep things from other namespaces
open if we get them. The chroot or equivalent binary is the one that needs
to ensure these kinds of issues don't exist if we care.

Speaking of we should put together a small test application probably similar
to chroot so people can access these features at least for testing.

> Small fix follows,
>
> thanks,
>
> C.

Ack. For the unshare fix below. Could you resend this one separately with
patch in the subject so Andrew sees it and picks up?

> From: Cedric Le Goater <clg@fr.ibm.com>
> Subject: ipc namespace : unshare fix
>
> Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
>
> ---
> kernel/fork.c | 3 ++-
> 1 file changed, 2 insertions(+), 1 deletion(-)
>
> Index: 2.6.17-rc6-mm2/kernel/fork.c
> ========== =======
> --- 2.6.17-rc6-mm2.orig/kernel/fork.c
> +++ 2.6.17-rc6-mm2/kernel/fork.c
> @@ -1599,7 +1599,8 @@ asmlinkage long sys_unshare(unsigned lon
>   /* Return -EINVAL for all unsupported flags */
>   err = -EINVAL;
>   if (unshare_flags & ~(CLONE_THREAD|CLONE_FS|CLONE_NEWNS|CLONE_SIGHAND|
> -   CLONE_VM|CLONE_FILES|CLONE_SYSVSEM|CLONE_NEWUTS))
> +   CLONE_VM|CLONE_FILES|CLONE_SYSVSEM|
> +   CLONE_NEWUTS|CLONE_NEWIPC))

> goto bad_unshare_out;
>
> if ((err = unshare_thread(unshare_flags)))

Subject: Re: [PATCH 2/6] IPC namespace - utils
Posted by Cedric Le Goater on Mon, 12 Jun 2006 21:05:34 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Cedric Le Goater <clg@fr.ibm.com> writes:
>
>> I've used the ipc namespace patchset in rc6-mm2. Thanks for putting this
>> together, it works pretty well ! A few questions when we clone :
>>
>> * We should do something close to what exit_sem() already does to clear the
>> sem_undo list from the task doing the clone() or unshare().
>
> Possibly which case are you trying to prevent?

task records a list of struct sem_undo each containing a semaphore id. When
we unshare ipc namespace, we break the 'reference' between the semaphore id
and the struct sem_array because the struct sem_array are cleared and freed
in the new namespace. When the task exit, that inconstency could lead to
unexpected results in exit_sem(), task locks, BUG_ON, etc. Nope ?

>> * I don't like the idea of being able to unshare the ipc namespace and keep
>> some shared memory from the previous ipc namespace mapped in the process mm.
>> Should we forbid the unshare ?
>
> No. As long as the code handles that case properly we should be fine.

what is the proper way to handle that case ? the current patchset is not
protected : a process can be in one ipc namespace and use a shared segment
from a previous ipc namespace. This situation is not desirable in a
migration scenario. May be asking too much for the moment ... and I agree
this can be fixed by the way namespaces are created.

> As a general principle we should be able to keep things from other namespaces
> open if we get them. The chroot or equivalent binary is the one that needs
> to ensure these kinds of issues don't exist if we care.
>
> Speaking of we should put together a small test application probably similar
> to chroot so people can access these features at least for testing.

are you thinking about a command unshare()ing each namespace or some kind
of create_nsproxy ?

> Ack. For the unshare fix below. Could you resend this one separately with
> patch in the subject so Andrew sees it and picks up?

done.

thanks,

c.

Subject: Re: [PATCH 2/6] IPC namespace - utils
Posted by ebiederm on Mon, 12 Jun 2006 21:49:03 GMT
View Forum Message <> Reply to Message

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

> Eric W. Biederman wrote:
>> Cedric Le Goater <clg@fr.ibm.com> writes:
>>
>>> I've used the ipc namespace patchset in rc6-mm2. Thanks for putting this
>>> together, it works pretty well ! A few questions when we clone :
>>>
>>> * We should do something close to what exit_sem() already does to clear the
>>> sem_undo list from the task doing the clone() or unshare().
>>
>> Possibly which case are you trying to prevent?
>
> task records a list of struct sem_undo each containing a semaphore id. When
> we unshare ipc namespace, we break the 'reference' between the semaphore id
> and the struct sem_array because the struct sem_array are cleared and freed
> in the new namespace. When the task exit, that inconstency could lead to
> unexpected results in exit_sem(), task locks, BUG_ON, etc. Nope ?

Agreed. Hmm. I bet I didn't see this one earlier because it is specific
to the unshare case. In this case I guess we should either deny the unshare
or simply undo all of the semaphores. Because we will never be able to
talk to them again.

Thinking about this some more we need to unsharing the semaphore undo semantics
when we create a new instances of the sysvipc namespace. Which means that
until that piece is implemented we can't unshare the sysvipc namespace.

But we clearly need the check in check_unshare_flags and the start of copy_process.

>>> * I don't like the idea of being able to unshare the ipc namespace and keep
>>> some shared memory from the previous ipc namespace mapped in the process mm.
>>> Should we forbid the unshare ?

>>
>> No. As long as the code handles that case properly we should be fine.
>
> what is the proper way to handle that case ? the current patchset is not
> protected : a process can be in one ipc namespace and use a shared segment
> from a previous ipc namespace. This situation is not desirable in a
> migration scenario. May be asking too much for the moment ... and I agree
> this can be fixed by the way namespaces are created.

As long as the appropriate reference counting happens it shouldn't be
a problem. We obviously can't use the sysvipc name of the shm area
but mmap and reads and writes should continue to work.

>> As a general principle we should be able to keep things from other namespaces
>> open if we get them. The chroot or equivalent binary is the one that needs
>> to ensure these kinds of issues don't exist if we care.
>>
>> Speaking of we should put together a small test application probably similar
>> to chroot so people can access these features at least for testing.
>
> are you thinking about a command unshare()ing each namespace or some kind
> of create_nsproxy ?

A little user space program like chroot. That takes a flag of which
namespaces not to share. I have one around somewhere. Just enough
of something that these interfaces can be exercised from userspace.

Eric

Subject: Re: [PATCH] IPC namespace
Posted by ebiederm on Tue, 13 Jun 2006 02:44:52 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@openvz.org> writes:

> The patches in this thread add IPC namespace functionality
> additionally to already included in -mm tree UTS namespace.
>
> This patch set allows to unshare IPCs and have a private set
> of IPC objects (sem, shm, msg) inside namespace. Basically, it is another
> building block of containers functionality.
>
> Tested with LTP inside namespaces.
>
> Signed-Off-By: Pavel Emelianiov <xemul@openvz.org>
> Signed-Off-By: Kirill Korotaev <dev@openvz.org>
>

> P.S. patches are against linux-2.6.17-rc6-mm1

Minor nit. These patches are not git-bisect safe.
So if you have to apply them all to get a kernel
that builds.

Anyone trying to narrow down breakage is likely to land
in the middle and hit a compile error.

Eric

Subject: Re: [PATCH] IPC namespace
Posted by dev on Tue, 13 Jun 2006 16:41:36 GMT
View Forum Message <> Reply to Message

>>P.S. patches are against linux-2.6.17-rc6-mm1
>
>
> Minor nit. These patches are not git-bisect safe.
> So if you have to apply them all to get a kernel
> that builds.
>
> Anyone trying to narrow down breakage is likely to land
> in the middle and hit a compile error.

Yes, these patches are hard to split (like with utsname...)
I splitted them for easier review mostly...

Kirill

Subject: Re: [PATCH] IPC namespace
Posted by ebiederm on Tue, 13 Jun 2006 17:01:21 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

>>>P.S. patches are against linux-2.6.17-rc6-mm1
>> Minor nit. These patches are not git-bisect safe.
>> So if you have to apply them all to get a kernel
>> that builds.
>> Anyone trying to narrow down breakage is likely to land
>> in the middle and hit a compile error.
>
> Yes, these patches are hard to split (like with utsname...)
> I splitted them for easier review mostly...

Reasonable.

I'm slowly making my way through reviewing them.

Eric

Subject: Re: [PATCH 2/6] IPC namespace - utils
Posted by Cedric Le Goater on Tue, 13 Jun 2006 21:17:44 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:

>> task records a list of struct sem_undo each containing a semaphore id. When
>> we unshare ipc namespace, we break the 'reference' between the semaphore id
>> and the struct sem_array because the struct sem_array are cleared and freed
>> in the new namespace. When the task exit, that inconstency could lead to
>> unexpected results in exit_sem(), task locks, BUG_ON, etc. Nope ?
>
> Agreed. Hmm. I bet I didn't see this one earlier because it is specific
> to the unshare case. In this case I guess we should either deny the unshare
> or simply undo all of the semaphores. Because we will never be able to
> talk to them again.

So aren't we reaching the unshare() limits ? Shouldn't we be using the
exec() principle for the sysvipc namespace ? clear it all and start from
scratch.

> Thinking about this some more we need to unsharing the semaphore undo semantics
> when we create a new instances of the sysvipc namespace. Which means that
> until that piece is implemented we can't unshare the sysvipc namespace.

no big issue I think. exit_sem() does it already. it would end up coding
the yet unsupported unshare_semundo().

> But we clearly need the check in check_unshare_flags and the start of copy_process.

Yes. CLONE_SYSVSEM and CLONE_NEWIPC overlap in some ways.

>>>> * I don't like the idea of being able to unshare the ipc namespace and keep
>>>> some shared memory from the previous ipc namespace mapped in the process mm.
>>>> Should we forbid the unshare ?
>>> No. As long as the code handles that case properly we should be fine.
>> what is the proper way to handle that case ? the current patchset is not
>> protected : a process can be in one ipc namespace and use a shared segment
>> from a previous ipc namespace. This situation is not desirable in a
>> migration scenario. May be asking too much for the moment ... and I agree

>> this can be fixed by the way namespaces are created.
>
> As long as the appropriate reference counting happens it shouldn't be
> a problem. We obviously can't use the sysvipc name of the shm area
> but mmap and reads and writes should continue to work.

in that case, namespace ids are protected but namespace objects aren't. I
expect a higher level object (container) making sure this is consistent.

C.

Subject: Re: [PATCH 2/6] IPC namespace - utils
Posted by dev on Wed, 14 Jun 2006 11:14:49 GMT
View Forum Message <> Reply to Message

>>Agreed. Hmm. I bet I didn't see this one earlier because it is specific
>>to the unshare case. In this case I guess we should either deny the unshare
>>or simply undo all of the semaphores. Because we will never be able to
>>talk to them again.
>
> So aren't we reaching the unshare() limits ? Shouldn't we be using the
> exec() principle for the sysvipc namespace ? clear it all and start from
> scratch.
there will be more such issues with more complex namespaces. That's why
I proposed to use containers.
Any way, right now, I don't think this should be urgently fixed as there
are many other ways to crash the node when you are a root.
The rule is simple - the process changing the namespace should be
simple, w/o IPCs. For more complex namespaces (e.g. resource management
namespaces) a fork() can be required after unshare().

>>Thinking about this some more we need to unsharing the semaphore undo semantics
>>when we create a new instances of the sysvipc namespace. Which means that
>>until that piece is implemented we can't unshare the sysvipc namespace.
>
>
> no big issue I think. exit_sem() does it already. it would end up coding
> the yet unsupported unshare_semundo().
do we need really it?
my point is that while with IPCs it maybe quite easy since semundo code
already exists it can be still very hard for other namespaces which do
not track its objects associated with the task.

BTW, do we have the same problem with shm? What will happen if the task
having shm segment will change it's IPC namespace? Besides the possible
crash/stability bugs there can be more interesting effects, for example:

"user will be able to do operations on existing objects with new namespace limitations?"...

Thanks,
Kirill