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) {
```