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;
```