Subject: [PATCH 2.6.24-rc8-mm1 00/15] IPC: code rewrite + new functionalities
Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:29 GMT
View Forum Message <> Reply to Message

Hi,

Here is a patchset about the IPC, which proposes to consolidate some part of the existing code and to add some functionalities.

* Patches 1 to 8 don't change the existing behavior, but propose to rewrite some parts of the existing code. In fact, the three kinds of IPC (semaphores, message queues and shared memory) have some common commands (IPC_SET, IPC_RMID, etc...) but they are mainly handled in three different ways. These patches propose to consolidate this, by handling these commands the same way and try to use, as much as possible, some common code. This should increase readability and maintainability of the code, making them probably good candidate for the -mm tree, I think.

* Patches 9 to 15 propose to add some functionalities, and thus are submitted here for RFC, about both the interest and their implementation.
These functionalities are:
   - Two new control-commands:
 . IPC_SETID: to change an IPC's id.
 . IPC_SETALL: behaves as IPC_SET, except that it also sets all time
   and pid values)
   - add a /proc/<pid>/semundo file to read and write the undo values of
some semaphores for a given process.

As the namespaces and the "containers" are being integrated in the kernel, these functionalities may be a first step to implement the checkpoint/restart of an application: in fact the existing API does not allow to specify or to change an ID when creating an IPC, when restarting an application, and the times/pids values of each IPCs are also altered. May be someone may find another interest about this ?

So again, comments are welcome.

Thanks.

--
Pierre

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 2.6.24-rc8-mm1 01/15] IPC/semaphores: code factorisation
Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:30 GMT
View Forum Message <> Reply to Message

From: Pierre Peiffer <pierre.peiffer@bull.net>

Trivial patch which adds some small locking functions and makes use of them
to factorize some part of code and makes it cleaner.

Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
Acked-by: Serge Hallyn <serue@us.ibm.com>

```
 ipc/sem.c |   61 +++++++++++++++++++++++++++++++------------------------------
 1 file changed, 31 insertions(+), 30 deletions(-)

Index: b/ipc/sem.c
===================================================================
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -181,6 +181,25 @@ static inline struct sem_array *sem_lock
  return container_of(ipcp, struct sem_array, sem_perm);
 }

+static inline void sem_lock_and_putref(struct sem_array *sma)
+{
+ ipc_lock_by_ptr(&sma->sem_perm);
+ ipc_rcu_putref(sma);
+}
+
+static inline void sem_getref_and_unlock(struct sem_array *sma)
+{
+ ipc_rcu_getref(sma);
+ ipc_unlock(&(sma)->sem_perm);
+}
+
+static inline void sem_putref(struct sem_array *sma)
+{
+ ipc_lock_by_ptr(&sma->sem_perm);
+ ipc_rcu_putref(sma);
+ ipc_unlock(&(sma)->sem_perm);
+}
+
 static inline void sem_rmid(struct ipc_namespace *ns, struct sem_array *s)
 {
  ipc_rmid(&sem_ids(ns), &s->sem_perm);
@@ -700,19 +719,15 @@ static int semctl_main(struct ipc_namesp
   int i;
   if(nsems > SEMMSL_FAST) {
-    ipc_rcu_getref(sma);
-    sem_unlock(sma);
+    sem_getref_and_unlock(sma);

     sem_io = ipc_alloc(sizeof(ushort)*nsems);
     if(sem_io == NULL) {
-     ipc_lock_by_ptr(&sma->sem_perm);
-     ipc_rcu_putref(sma);
-     sem_unlock(sma);
+     sem_putref(sma);
      return -ENOMEM;
     }

-    ipc_lock_by_ptr(&sma->sem_perm);
-    ipc_rcu_putref(sma);
+    sem_lock_and_putref(sma);
     if (sma->sem_perm.deleted) {
      sem_unlock(sma);
      err = -EIDRM;
@@ -733,38 +748,30 @@ static int semctl_main(struct ipc_namesp
   int i;
   struct sem_undo *un;

-  ipc_rcu_getref(sma);
-  sem_unlock(sma);
+  sem_getref_and_unlock(sma);

   if(nsems > SEMMSL_FAST) {
     sem_io = ipc_alloc(sizeof(ushort)*nsems);
     if(sem_io == NULL) {
-     ipc_lock_by_ptr(&sma->sem_perm);
-     ipc_rcu_putref(sma);
-     sem_unlock(sma);
+     sem_putref(sma);
      return -ENOMEM;
     }
   }

   if (copy_from_user (sem_io, arg.array, nsems*sizeof(ushort))) {
-    ipc_lock_by_ptr(&sma->sem_perm);
-    ipc_rcu_putref(sma);
-    sem_unlock(sma);
+    sem_putref(sma);
     err = -EFAULT;
     goto out_free;
   }
   for (i = 0; i < nsems; i++) {
    if (sem_io[i] > SEMVMX) {
-    ipc_lock_by_ptr(&sma->sem_perm);
-    ipc_rcu_putref(sma);
-    sem_unlock(sma);
+    sem_putref(sma);
     err = -ERANGE;
     goto out_free;
    }
   }
-  ipc_lock_by_ptr(&sma->sem_perm);
-  ipc_rcu_putref(sma);
+  sem_lock_and_putref(sma);
   if (sma->sem_perm.deleted) {
    sem_unlock(sma);
    err = -EIDRM;
@@ -1044,14 +1051,11 @@ static struct sem_undo *find_undo(struct
   return ERR_PTR(PTR_ERR(sma));

  nsems = sma->sem_nsems;
- ipc_rcu_getref(sma);
- sem_unlock(sma);
+ sem_getref_and_unlock(sma);

  new = kzalloc(sizeof(struct sem_undo) + sizeof(short)*nsems, GFP_KERNEL);
  if (!new) {
-  ipc_lock_by_ptr(&sma->sem_perm);
-  ipc_rcu_putref(sma);
-  sem_unlock(sma);
+  sem_putref(sma);
   return ERR_PTR(-ENOMEM);
  }
  new->semadj = (short *) &new[1];
@@ -1062,13 +1066,10 @@ static struct sem_undo *find_undo(struct
  if (un) {
   spin_unlock(&ulp->lock);
   kfree(new);
-  ipc_lock_by_ptr(&sma->sem_perm);
-  ipc_rcu_putref(sma);
-  sem_unlock(sma);
+  sem_putref(sma);
   goto out;
  }
- ipc_lock_by_ptr(&sma->sem_perm);
- ipc_rcu_putref(sma);
+ sem_lock_and_putref(sma);
  if (sma->sem_perm.deleted) {
   sem_unlock(sma);
```

```
	spin_unlock(&ulp->lock);
```

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 2.6.24-rc8-mm1 02/15] IPC/shared memory: introduce shmctl_down
Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:31 GMT
View Forum Message <> Reply to Message

From: Pierre Peiffer <pierre.peiffer@bull.net>

Currently, the way the different commands are handled in sys_shmctl
introduces some duplicated code.
This patch introduces the shmctl_down function to handle all the commands
requiring the rwmutex to be taken in write mode (ie IPC_SET and IPC_RMID
for now). It is the equivalent function of semctl_down for shared
memory.

This removes some duplicated code for handling these both commands
and harmonizes the way they are handled among all IPCs.

Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
Acked-by: Serge Hallyn <serue@us.ibm.com>

```
 ipc/shm.c |  160 +++++++++++++++++++++++++++----------------------------
 1 file changed, 72 insertions(+), 88 deletions(-)

Index: b/ipc/shm.c
===================================================================
--- a/ipc/shm.c
+++ b/ipc/shm.c
@@ -625,10 +625,78 @@ static void shm_get_stat(struct ipc_name
 	}
 }

-asmlinkage long sys_shmctl (int shmid, int cmd, struct shmid_ds __user *buf)
+/*
+ * This function handles some shmctl commands which require the rw_mutex
+ * to be held in write mode.
+ * NOTE: no locks must be held, the rw_mutex is taken inside this function.
+ */
```

```
+static int shmctl_down(struct ipc_namespace *ns, int shmid, int cmd,
+        struct shmid_ds __user *buf, int version)
 {
+ struct kern_ipc_perm *ipcp;
  struct shm_setbuf setbuf;
  struct shmid_kernel *shp;
+ int err;
+
+ if (cmd == IPC_SET) {
+  if (copy_shmid_from_user(&setbuf, buf, version))
+   return -EFAULT;
+ }
+
+ down_write(&shm_ids(ns).rw_mutex);
+ shp = shm_lock_check_down(ns, shmid);
+ if (IS_ERR(shp)) {
+  err = PTR_ERR(shp);
+  goto out_up;
+ }
+
+ ipcp = &shp->shm_perm;
+
+ err = audit_ipc_obj(ipcp);
+ if (err)
+  goto out_unlock;
+
+ if (cmd == IPC_SET) {
+  err = audit_ipc_set_perm(0, setbuf.uid,
+      setbuf.gid, setbuf.mode);
+  if (err)
+   goto out_unlock;
+ }
+
+ if (current->euid != ipcp->uid &&
+     current->euid != ipcp->cuid &&
+     !capable(CAP_SYS_ADMIN)) {
+  err = -EPERM;
+  goto out_unlock;
+ }
+
+ err = security_shm_shmctl(shp, cmd);
+ if (err)
+  goto out_unlock;
+ switch (cmd) {
+ case IPC_RMID:
+  do_shm_rmid(ns, ipcp);
+  goto out_up;
+ case IPC_SET:
```

```
+  ipcp->uid = setbuf.uid;
+  ipcp->gid = setbuf.gid;
+  ipcp->mode = (ipcp->mode & ~S_IRWXUGO)
+   | (setbuf.mode & S_IRWXUGO);
+  shp->shm_ctim = get_seconds();
+  break;
+ default:
+  err = -EINVAL;
+ }
+out_unlock:
+ shm_unlock(shp);
+out_up:
+ up_write(&shm_ids(ns).rw_mutex);
+ return err;
+}
+
+asmlinkage long sys_shmctl(int shmid, int cmd, struct shmid_ds __user *buf)
+{
+ struct shmid_kernel *shp;
  int err, version;
  struct ipc_namespace *ns;

@@ -784,97 +852,13 @@ asmlinkage long sys_shmctl (int shmid, i
   goto out;
  }
  case IPC_RMID:
- {
-  /*
-   * We cannot simply remove the file. The SVID states
-   * that the block remains until the last person
-   * detaches from it, then is deleted. A shmat() on
-   * an RMID segment is legal in older Linux and if
-   * we change it apps break...
-   *
-   * Instead we set a destroyed flag, and then blow
-   * the name away when the usage hits zero.
-   */
-  down_write(&shm_ids(ns).rw_mutex);
-  shp = shm_lock_check_down(ns, shmid);
-  if (IS_ERR(shp)) {
-   err = PTR_ERR(shp);
-   goto out_up;
-  }
-
-  err = audit_ipc_obj(&(shp->shm_perm));
-  if (err)
-   goto out_unlock_up;
-
```

```
-  if (current->euid != shp->shm_perm.uid &&
-      current->euid != shp->shm_perm.cuid &&
-      !capable(CAP_SYS_ADMIN)) {
-   err=-EPERM;
-   goto out_unlock_up;
-  }
-
-  err = security_shm_shmctl(shp, cmd);
-  if (err)
-   goto out_unlock_up;
-
-  do_shm_rmid(ns, &shp->shm_perm);
-  up_write(&shm_ids(ns).rw_mutex);
-  goto out;
- }
-
  case IPC_SET:
- {
-  if (!buf) {
-   err = -EFAULT;
-   goto out;
-  }
-
-  if (copy_shmid_from_user (&setbuf, buf, version)) {
-   err = -EFAULT;
-   goto out;
-  }
-  down_write(&shm_ids(ns).rw_mutex);
-  shp = shm_lock_check_down(ns, shmid);
-  if (IS_ERR(shp)) {
-   err = PTR_ERR(shp);
-   goto out_up;
-  }
-  err = audit_ipc_obj(&(shp->shm_perm));
-  if (err)
-   goto out_unlock_up;
-  err = audit_ipc_set_perm(0, setbuf.uid, setbuf.gid, setbuf.mode);
-  if (err)
-   goto out_unlock_up;
-  err=-EPERM;
-  if (current->euid != shp->shm_perm.uid &&
-      current->euid != shp->shm_perm.cuid &&
-      !capable(CAP_SYS_ADMIN)) {
-   goto out_unlock_up;
-  }
-
-  err = security_shm_shmctl(shp, cmd);
-  if (err)
-   goto out_unlock_up;
-
-  shp->shm_perm.uid = setbuf.uid;
-  shp->shm_perm.gid = setbuf.gid;
-  shp->shm_perm.mode = (shp->shm_perm.mode & ~S_IRWXUGO)
-   | (setbuf.mode & S_IRWXUGO);
-  shp->shm_ctim = get_seconds();
-  break;
- }
-
+  err = shmctl_down(ns, shmid, cmd, buf, version);
+  return err;
  default:
-  err = -EINVAL;
-  goto out;
+  return -EINVAL;
  }

- err = 0;
-out_unlock_up:
- shm_unlock(shp);
-out_up:
- up_write(&shm_ids(ns).rw_mutex);
- goto out;
 out_unlock:
  shm_unlock(shp);
 out:
```

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 2.6.24-rc8-mm1 03/15] IPC/message queues: introduce msgctl_down

Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:32 GMT

View Forum Message <> Reply to Message

From: Pierre Peiffer <pierre.peiffer@bull.net>

Currently, sys_msgctl is not easy to read.
This patch tries to improve that by introducing the msgctl_down function
to handle all commands requiring the rwmutex to be taken in write mode
(ie IPC_SET and IPC_RMID for now). It is the equivalent function of
semctl_down for message queues.

This greatly changes the readability of sys_msgctl and also harmonizes
the way these commands are handled among all IPCs.

Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
Acked-by: Serge Hallyn <serue@us.ibm.com>

```
 ipc/msg.c |  162 +++++++++++++++++++++++++++++++++++-----------------------------------
 1 file changed, 89 insertions(+), 73 deletions(-)

Index: b/ipc/msg.c
===================================================================
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -399,10 +399,95 @@ copy_msqid_from_user(struct msq_setbuf *
  }
 }

-asmlinkage long sys_msgctl(int msqid, int cmd, struct msqid_ds __user *buf)
+/*
+ * This function handles some msgctl commands which require the rw_mutex
+ * to be held in write mode.
+ * NOTE: no locks must be held, the rw_mutex is taken inside this function.
+ */
+static int msgctl_down(struct ipc_namespace *ns, int msqid, int cmd,
+         struct msqid_ds __user *buf, int version)
 {
  struct kern_ipc_perm *ipcp;
- struct msq_setbuf uninitialized_var(setbuf);
+ struct msq_setbuf setbuf;
+ struct msg_queue *msq;
+ int err;
+
+ if (cmd == IPC_SET) {
+  if (copy_msqid_from_user(&setbuf, buf, version))
+   return -EFAULT;
+ }
+
+ down_write(&msg_ids(ns).rw_mutex);
+ msq = msg_lock_check_down(ns, msqid);
+ if (IS_ERR(msq)) {
+  err = PTR_ERR(msq);
+  goto out_up;
+ }
+
+ ipcp = &msq->q_perm;
```

```
+
+ err = audit_ipc_obj(ipcp);
+ if (err)
+  goto out_unlock;
+
+ if (cmd == IPC_SET) {
+  err = audit_ipc_set_perm(setbuf.qbytes, setbuf.uid, setbuf.gid,
+      setbuf.mode);
+  if (err)
+   goto out_unlock;
+ }
+
+ if (current->euid != ipcp->cuid &&
+     current->euid != ipcp->uid &&
+     !capable(CAP_SYS_ADMIN)) {
+  /* We _could_ check for CAP_CHOWN above, but we don't */
+  err = -EPERM;
+  goto out_unlock;
+ }
+
+ err = security_msg_queue_msgctl(msq, cmd);
+ if (err)
+  goto out_unlock;
+
+ switch (cmd) {
+ case IPC_RMID:
+  freeque(ns, ipcp);
+  goto out_up;
+ case IPC_SET:
+  if (setbuf.qbytes > ns->msg_ctlmnb &&
+      !capable(CAP_SYS_RESOURCE)) {
+   err = -EPERM;
+   goto out_unlock;
+  }
+
+  msq->q_qbytes = setbuf.qbytes;
+
+  ipcp->uid = setbuf.uid;
+  ipcp->gid = setbuf.gid;
+  ipcp->mode = (ipcp->mode & ~S_IRWXUGO) |
+        (S_IRWXUGO & setbuf.mode);
+  msq->q_ctime = get_seconds();
+  /* sleeping receivers might be excluded by
+   * stricter permissions.
+   */
+  expunge_all(msq, -EAGAIN);
+  /* sleeping senders might be able to send
+   * due to a larger queue size.
+   */
+  ss_wakeup(&msq->q_senders, 0);
+  break;
+ default:
+  err = -EINVAL;
+ }
+out_unlock:
+ msg_unlock(msq);
+out_up:
+ up_write(&msg_ids(ns).rw_mutex);
+ return err;
+}
+
+asmlinkage long sys_msgctl(int msqid, int cmd, struct msqid_ds __user *buf)
+{
  struct msg_queue *msq;
  int err, version;
  struct ipc_namespace *ns;
@@ -498,82 +583,13 @@ asmlinkage long sys_msgctl(int msqid, in
   return success_return;
  }
  case IPC_SET:
-  if (!buf)
-   return -EFAULT;
-  if (copy_msqid_from_user(&setbuf, buf, version))
-   return -EFAULT;
-  break;
  case IPC_RMID:
-  break;
+  err = msgctl_down(ns, msqid, cmd, buf, version);
+  return err;
  default:
   return  -EINVAL;
  }

- down_write(&msg_ids(ns).rw_mutex);
- msq = msg_lock_check_down(ns, msqid);
- if (IS_ERR(msq)) {
-  err = PTR_ERR(msq);
-  goto out_up;
- }
-
- ipcp = &msq->q_perm;
-
- err = audit_ipc_obj(ipcp);
- if (err)
-  goto out_unlock_up;
- if (cmd == IPC_SET) {
```

```
-  err = audit_ipc_set_perm(setbuf.qbytes, setbuf.uid, setbuf.gid,
-      setbuf.mode);
-  if (err)
-   goto out_unlock_up;
- }
-
- err = -EPERM;
- if (current->euid != ipcp->cuid &&
-     current->euid != ipcp->uid && !capable(CAP_SYS_ADMIN))
-  /* We _could_ check for CAP_CHOWN above, but we don't */
-  goto out_unlock_up;
-
- err = security_msg_queue_msgctl(msq, cmd);
- if (err)
-  goto out_unlock_up;
-
- switch (cmd) {
- case IPC_SET:
- {
-  err = -EPERM;
-  if (setbuf.qbytes > ns->msg_ctlmnb && !capable(CAP_SYS_RESOURCE))
-   goto out_unlock_up;
-
-  msq->q_qbytes = setbuf.qbytes;
-
-  ipcp->uid = setbuf.uid;
-  ipcp->gid = setbuf.gid;
-  ipcp->mode = (ipcp->mode & ~S_IRWXUGO) |
-        (S_IRWXUGO & setbuf.mode);
-  msq->q_ctime = get_seconds();
-  /* sleeping receivers might be excluded by
-   * stricter permissions.
-   */
-  expunge_all(msq, -EAGAIN);
-  /* sleeping senders might be able to send
-   * due to a larger queue size.
-   */
-  ss_wakeup(&msq->q_senders, 0);
-  msg_unlock(msq);
-  break;
- }
- case IPC_RMID:
-  freeque(ns, &msq->q_perm);
-  break;
- }
- err = 0;
-out_up:
- up_write(&msg_ids(ns).rw_mutex);
- return err;
-out_unlock_up:
- msg_unlock(msq);
- goto out_up;
 out_unlock:
  msg_unlock(msq);
  return err;
```

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 2.6.24-rc8-mm1 04/15] IPC/semaphores: move the rwmutex handling inside semctl_down
Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:33 GMT

View Forum Message <> Reply to Message

From: Pierre Peiffer <pierre.peiffer@bull.net>

semctl_down is called with the rwmutex (the one which protects the
list of ipcs) taken in write mode.
This patch moves this rwmutex taken in write-mode inside semctl_down.
This has the advantages of reducing a little bit the window during
which this rwmutex is taken, clarifying sys_semctl, and finally of
having a coherent behaviour with [shm|msg]ctl_down

Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
Acked-by: Serge Hallyn <serue@us.ibm.com>

```
 ipc/sem.c |   24 +++++++++++++-----------
 1 file changed, 13 insertions(+), 11 deletions(-)

Index: b/ipc/sem.c
===================================================================
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -877,6 +877,11 @@ static inline unsigned long copy_semid_f
  }
 }

+/*
+ * This function handles some semctl commands which require the rw_mutex
+ * to be held in write mode.
+ * NOTE: no locks must be held, the rw_mutex is taken inside this function.
+ */
 static int semctl_down(struct ipc_namespace *ns, int semid, int semnum,
   int cmd, int version, union semun arg)
 {
@@ -889,9 +894,12 @@ static int semctl_down(struct ipc_namesp
   if(copy_semid_from_user (&setbuf, arg.buf, version))
    return -EFAULT;
  }
+ down_write(&sem_ids(ns).rw_mutex);
  sma = sem_lock_check_down(ns, semid);
- if (IS_ERR(sma))
-  return PTR_ERR(sma);
+ if (IS_ERR(sma)) {
+  err = PTR_ERR(sma);
+  goto out_up;
+ }

  ipcp = &sma->sem_perm;

@@ -917,26 +925,22 @@ static int semctl_down(struct ipc_namesp
  switch(cmd){
  case IPC_RMID:
   freeary(ns, ipcp);
-  err = 0;
-  break;
+  goto out_up;
  case IPC_SET:
   ipcp->uid = setbuf.uid;
   ipcp->gid = setbuf.gid;
   ipcp->mode = (ipcp->mode & ~S_IRWXUGO)
     | (setbuf.mode & S_IRWXUGO);
   sma->sem_ctime = get_seconds();
-  sem_unlock(sma);
-  err = 0;
   break;
  default:
-  sem_unlock(sma);
   err = -EINVAL;
-  break;
  }
- return err;

 out_unlock:
  sem_unlock(sma);
+out_up:
+ up_write(&sem_ids(ns).rw_mutex);
  return err;
 }

@@ -970,9 +974,7 @@ asmlinkage long sys_semctl (int semid, i
   return err;
  case IPC_RMID:
  case IPC_SET:
-  down_write(&sem_ids(ns).rw_mutex);
   err = semctl_down(ns,semid,semnum,cmd,version,arg);
-  up_write(&sem_ids(ns).rw_mutex);
   return err;
  default:
   return -EINVAL;
```

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 2.6.24-rc8-mm1 05/15] IPC/semaphores: remove one unused parameter from semctl_down()

Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:34 GMT

View Forum Message <> Reply to Message

From: Pierre Peiffer <pierre.peiffer@bull.net>

semctl_down() takes one unused parameter: semnum.
This patch proposes to get rid of it.

Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
Acked-by: Serge Hallyn <serue@us.ibm.com>

```
 ipc/sem.c |    6 +++---
 1 file changed, 3 insertions(+), 3 deletions(-)

Index: b/ipc/sem.c
===================================================================
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -882,8 +882,8 @@ static inline unsigned long copy_semid_f
  * to be held in write mode.
  * NOTE: no locks must be held, the rw_mutex is taken inside this function.
  */
-static int semctl_down(struct ipc_namespace *ns, int semid, int semnum,
-  int cmd, int version, union semun arg)
+static int semctl_down(struct ipc_namespace *ns, int semid,
+         int cmd, int version, union semun arg)
 {
  struct sem_array *sma;
  int err;
@@ -974,7 +974,7 @@ asmlinkage long sys_semctl (int semid, i
   return err;
  case IPC_RMID:
  case IPC_SET:
-  err = semctl_down(ns,semid,semnum,cmd,version,arg);
+  err = semctl_down(ns, semid, cmd, version, arg);
   return err;
  default:
   return -EINVAL;
```

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 2.6.24-rc8-mm1 06/15] IPC: get rid of the use *_setbuf structure.
Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:35 GMT

View Forum Message <> Reply to Message

From: Pierre Peiffer <pierre.peiffer@bull.net>

All IPCs make use of an intermetiate *_setbuf structure to handle the
IPC_SET command. This is not really needed and, moreover, it complicate
a little bit the code.

This patch get rid of the use of it and uses directly the semid64_ds/
msgid64_ds/shmid64_ds structure.

In addition of removing one struture declaration, it also simplifies
and improves a little bit the common 64-bits path.
Moreover, this will simplify the code for handling the IPC_SETALL
command provided in the next patch.

Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
Acked-by: Serge Hallyn <serue@us.ibm.com>

```
 ipc/msg.c |   51 ++++++++++++++++++-----------------------------------
 ipc/sem.c |   40 ++++++++++++++--------------------------
 ipc/shm.c |   41 ++++++++++++++---------------------------
 3 files changed, 46 insertions(+), 86 deletions(-)
```

```
Index: b/ipc/msg.c
===================================================================
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -351,31 +351,14 @@ copy_msqid_to_user(void __user *buf, str
  }
 }

-struct msq_setbuf {
- unsigned long qbytes;
- uid_t  uid;
- gid_t  gid;
- mode_t  mode;
-};
-
 static inline unsigned long
-copy_msqid_from_user(struct msq_setbuf *out, void __user *buf, int version)
+copy_msqid_from_user(struct msqid64_ds *out, void __user *buf, int version)
 {
  switch(version) {
  case IPC_64:
- {
-  struct msqid64_ds tbuf;
-
-  if (copy_from_user(&tbuf, buf, sizeof(tbuf)))
+  if (copy_from_user(out, buf, sizeof(*out)))
    return -EFAULT;
-
-  out->qbytes  = tbuf.msg_qbytes;
-  out->uid  = tbuf.msg_perm.uid;
-  out->gid  = tbuf.msg_perm.gid;
-  out->mode  = tbuf.msg_perm.mode;
-
   return 0;
- }
  case IPC_OLD:
  {
   struct msqid_ds tbuf_old;
@@ -383,14 +366,14 @@ copy_msqid_from_user(struct msq_setbuf *
   if (copy_from_user(&tbuf_old, buf, sizeof(tbuf_old)))
    return -EFAULT;

-  out->uid  = tbuf_old.msg_perm.uid;
-  out->gid  = tbuf_old.msg_perm.gid;
-  out->mode  = tbuf_old.msg_perm.mode;
+  out->msg_perm.uid       = tbuf_old.msg_perm.uid;
+  out->msg_perm.gid       = tbuf_old.msg_perm.gid;
+  out->msg_perm.mode      = tbuf_old.msg_perm.mode;

   if (tbuf_old.msg_qbytes == 0)
-   out->qbytes = tbuf_old.msg_lqbytes;
+   out->msg_qbytes = tbuf_old.msg_lqbytes;
   else
-   out->qbytes = tbuf_old.msg_qbytes;
+   out->msg_qbytes = tbuf_old.msg_qbytes;

   return 0;
  }
@@ -408,12 +391,12 @@ static int msgctl_down(struct ipc_namesp
         struct msqid_ds __user *buf, int version)
 {
  struct kern_ipc_perm *ipcp;
- struct msq_setbuf setbuf;
+ struct msqid64_ds msqid64;
  struct msg_queue *msq;
  int err;

  if (cmd == IPC_SET) {
-  if (copy_msqid_from_user(&setbuf, buf, version))
+  if (copy_msqid_from_user(&msqid64, buf, version))
    return -EFAULT;
  }

@@ -431,8 +414,10 @@ static int msgctl_down(struct ipc_namesp
   goto out_unlock;

  if (cmd == IPC_SET) {
-  err = audit_ipc_set_perm(setbuf.qbytes, setbuf.uid, setbuf.gid,
-      setbuf.mode);
+  err = audit_ipc_set_perm(msqid64.msg_qbytes,
+      msqid64.msg_perm.uid,
+      msqid64.msg_perm.gid,
+      msqid64.msg_perm.mode);
   if (err)
    goto out_unlock;
  }
@@ -454,18 +439,18 @@ static int msgctl_down(struct ipc_namesp
   freeque(ns, ipcp);
   goto out_up;
  case IPC_SET:
-  if (setbuf.qbytes > ns->msg_ctlmnb &&
+  if (msqid64.msg_qbytes > ns->msg_ctlmnb &&
      !capable(CAP_SYS_RESOURCE)) {
    err = -EPERM;
    goto out_unlock;
   }

-  msq->q_qbytes = setbuf.qbytes;
+  msq->q_qbytes = msqid64.msg_qbytes;

-  ipcp->uid = setbuf.uid;
-  ipcp->gid = setbuf.gid;
+  ipcp->uid = msqid64.msg_perm.uid;
+  ipcp->gid = msqid64.msg_perm.gid;
   ipcp->mode = (ipcp->mode & ~S_IRWXUGO) |
-       (S_IRWXUGO & setbuf.mode);
+       (S_IRWXUGO & msqid64.msg_perm.mode);
   msq->q_ctime = get_seconds();
   /* sleeping receivers might be excluded by
    * stricter permissions.
Index: b/ipc/sem.c
===================================================================
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -837,28 +837,14 @@ out_free:
  return err;
 }

-struct sem_setbuf {
- uid_t uid;
- gid_t gid;
- mode_t mode;
-};
-
-static inline unsigned long copy_semid_from_user(struct sem_setbuf *out, void __user *buf, int version)
+static inline unsigned long
+copy_semid_from_user(struct semid64_ds *out, void __user *buf, int version)
 {
  switch(version) {
  case IPC_64:
-    {
-  struct semid64_ds tbuf;
-
-  if(copy_from_user(&tbuf, buf, sizeof(tbuf)))
+  if (copy_from_user(out, buf, sizeof(*out)))
    return -EFAULT;
-
-  out->uid = tbuf.sem_perm.uid;
-  out->gid = tbuf.sem_perm.gid;
-  out->mode = tbuf.sem_perm.mode;
-
   return 0;
```

```
-    }
  case IPC_OLD:
      {
   struct semid_ds tbuf_old;
@@ -866,9 +852,9 @@ static inline unsigned long copy_semid_f
   if(copy_from_user(&tbuf_old, buf, sizeof(tbuf_old)))
    return -EFAULT;

-  out->uid = tbuf_old.sem_perm.uid;
-  out->gid = tbuf_old.sem_perm.gid;
-  out->mode = tbuf_old.sem_perm.mode;
+  out->sem_perm.uid = tbuf_old.sem_perm.uid;
+  out->sem_perm.gid = tbuf_old.sem_perm.gid;
+  out->sem_perm.mode = tbuf_old.sem_perm.mode;

   return 0;
      }
@@ -887,11 +873,11 @@ static int semctl_down(struct ipc_namesp
 {
  struct sem_array *sma;
  int err;
- struct sem_setbuf uninitialized_var(setbuf);
+ struct semid64_ds semid64;
  struct kern_ipc_perm *ipcp;

  if(cmd == IPC_SET) {
-  if(copy_semid_from_user (&setbuf, arg.buf, version))
+  if (copy_semid_from_user(&semid64, arg.buf, version))
    return -EFAULT;
  }
  down_write(&sem_ids(ns).rw_mutex);
@@ -908,7 +894,9 @@ static int semctl_down(struct ipc_namesp
   goto out_unlock;

  if (cmd == IPC_SET) {
-  err = audit_ipc_set_perm(0, setbuf.uid, setbuf.gid, setbuf.mode);
+  err = audit_ipc_set_perm(0, semid64.sem_perm.uid,
+      semid64.sem_perm.gid,
+      semid64.sem_perm.mode);
   if (err)
    goto out_unlock;
  }
@@ -927,10 +915,10 @@ static int semctl_down(struct ipc_namesp
   freeary(ns, ipcp);
   goto out_up;
  case IPC_SET:
-  ipcp->uid = setbuf.uid;
-  ipcp->gid = setbuf.gid;
```

```
+  ipcp->uid = semid64.sem_perm.uid;
+  ipcp->gid = semid64.sem_perm.gid;
   ipcp->mode = (ipcp->mode & ~S_IRWXUGO)
-    | (setbuf.mode & S_IRWXUGO);
+    | (semid64.sem_perm.mode & S_IRWXUGO);
   sma->sem_ctime = get_seconds();
   break;
  default:
Index: b/ipc/shm.c
===================================================================
--- a/ipc/shm.c
+++ b/ipc/shm.c
@@ -520,28 +520,14 @@ static inline unsigned long copy_shmid_t
  }
 }

-struct shm_setbuf {
- uid_t uid;
- gid_t gid;
- mode_t mode;
-};
-
-static inline unsigned long copy_shmid_from_user(struct shm_setbuf *out, void __user *buf, int version)
+static inline unsigned long
+copy_shmid_from_user(struct shmid64_ds *out, void __user *buf, int version)
 {
  switch(version) {
  case IPC_64:
-    {
-  struct shmid64_ds tbuf;
-
-  if (copy_from_user(&tbuf, buf, sizeof(tbuf)))
+  if (copy_from_user(out, buf, sizeof(*out)))
    return -EFAULT;
-
-  out->uid = tbuf.shm_perm.uid;
-  out->gid = tbuf.shm_perm.gid;
-  out->mode = tbuf.shm_perm.mode;
-
   return 0;
-    }
  case IPC_OLD:
     {
   struct shmid_ds tbuf_old;
@@ -549,9 +535,9 @@ static inline unsigned long copy_shmid_f
   if (copy_from_user(&tbuf_old, buf, sizeof(tbuf_old)))
    return -EFAULT;
```

```
-  out->uid = tbuf_old.shm_perm.uid;
-  out->gid = tbuf_old.shm_perm.gid;
-  out->mode = tbuf_old.shm_perm.mode;
+  out->shm_perm.uid = tbuf_old.shm_perm.uid;
+  out->shm_perm.gid = tbuf_old.shm_perm.gid;
+  out->shm_perm.mode = tbuf_old.shm_perm.mode;

   return 0;
      }
@@ -634,12 +620,12 @@ static int shmctl_down(struct ipc_namesp
          struct shmid_ds __user *buf, int version)
 {
  struct kern_ipc_perm *ipcp;
- struct shm_setbuf setbuf;
+ struct shmid64_ds shmid64;
  struct shmid_kernel *shp;
  int err;

  if (cmd == IPC_SET) {
-  if (copy_shmid_from_user(&setbuf, buf, version))
+  if (copy_shmid_from_user(&shmid64, buf, version))
    return -EFAULT;
  }

@@ -657,8 +643,9 @@ static int shmctl_down(struct ipc_namesp
   goto out_unlock;

  if (cmd == IPC_SET) {
-  err = audit_ipc_set_perm(0, setbuf.uid,
-      setbuf.gid, setbuf.mode);
+  err = audit_ipc_set_perm(0, shmid64.shm_perm.uid,
+      shmid64.shm_perm.gid,
+      shmid64.shm_perm.mode);
   if (err)
    goto out_unlock;
  }
@@ -678,10 +665,10 @@ static int shmctl_down(struct ipc_namesp
   do_shm_rmid(ns, ipcp);
   goto out_up;
  case IPC_SET:
-  ipcp->uid = setbuf.uid;
-  ipcp->gid = setbuf.gid;
+  ipcp->uid = shmid64.shm_perm.uid;
+  ipcp->gid = shmid64.shm_perm.gid;
   ipcp->mode = (ipcp->mode & ~S_IRWXUGO)
-   | (setbuf.mode & S_IRWXUGO);
+   | (shmid64.shm_perm.mode & S_IRWXUGO);
```

```
   shp->shm_ctim = get_seconds();
   break;
  default:
```

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 2.6.24-rc8-mm1 07/15] IPC: introduce ipc_update_perm()
Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:36 GMT

View Forum Message <> Reply to Message

From: Pierre Peiffer <pierre.peiffer@bull.net>

The IPC_SET command performs the same permission setting for all IPCs.
This patch introduces a common ipc_update_perm() function to update these
permissions and makes use of it for all IPCs.

Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
Acked-by: Serge Hallyn <serue@us.ibm.com>

```
 ipc/msg.c  |    5 +----
 ipc/sem.c  |    5 +----
 ipc/shm.c  |    5 +----
 ipc/util.c |   13 +++++++++++++
 ipc/util.h |    1 +
 5 files changed, 17 insertions(+), 12 deletions(-)
```

```
Index: b/ipc/msg.c
===================================================================
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -447,10 +447,7 @@ static int msgctl_down(struct ipc_namesp

   msq->q_qbytes = msqid64.msg_qbytes;

-  ipcp->uid = msqid64.msg_perm.uid;
-  ipcp->gid = msqid64.msg_perm.gid;
-  ipcp->mode = (ipcp->mode & ~S_IRWXUGO) |
-        (S_IRWXUGO & msqid64.msg_perm.mode);
+  ipc_update_perm(&msqid64.msg_perm, ipcp);
   msq->q_ctime = get_seconds();
   /* sleeping receivers might be excluded by
   * stricter permissions.
Index: b/ipc/sem.c
===================================================================
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -915,10 +915,7 @@ static int semctl_down(struct ipc_namesp
   freeary(ns, ipcp);
   goto out_up;
  case IPC_SET:
-  ipcp->uid = semid64.sem_perm.uid;
-  ipcp->gid = semid64.sem_perm.gid;
-  ipcp->mode = (ipcp->mode & ~S_IRWXUGO)
-    | (semid64.sem_perm.mode & S_IRWXUGO);
+  ipc_update_perm(&semid64.sem_perm, ipcp);
   sma->sem_ctime = get_seconds();
   break;
  default:
Index: b/ipc/shm.c
===================================================================
--- a/ipc/shm.c
+++ b/ipc/shm.c
@@ -665,10 +665,7 @@ static int shmctl_down(struct ipc_namesp
   do_shm_rmid(ns, ipcp);
   goto out_up;
  case IPC_SET:
-  ipcp->uid = shmid64.shm_perm.uid;
-  ipcp->gid = shmid64.shm_perm.gid;
-  ipcp->mode = (ipcp->mode & ~S_IRWXUGO)
-   | (shmid64.shm_perm.mode & S_IRWXUGO);
+  ipc_update_perm(&shmid64.shm_perm, ipcp);
   shp->shm_ctim = get_seconds();
   break;
  default:
Index: b/ipc/util.c
===================================================================
--- a/ipc/util.c
+++ b/ipc/util.c
@@ -761,6 +761,19 @@ int ipcget(struct ipc_namespace *ns, str
   return ipcget_public(ns, ids, ops, params);
 }

+/**
+ * ipc_update_perm - update the permissions of an IPC.
+ * @in:  the permission given as input.
+ * @out: the permission of the ipc to set.
+ */
+void ipc_update_perm(struct ipc64_perm *in, struct kern_ipc_perm *out)
+{
```

```
+ out->uid = in->uid;
+ out->gid = in->gid;
+ out->mode = (out->mode & ~S_IRWXUGO)
+  | (in->mode & S_IRWXUGO);
+}
+
 #ifdef __ARCH_WANT_IPC_PARSE_VERSION
```

```
Index: b/ipc/util.h
===================================================================
--- a/ipc/util.h
+++ b/ipc/util.h
@@ -112,6 +112,7 @@ struct kern_ipc_perm *ipc_lock(struct ip

 void kernel_to_ipc64_perm(struct kern_ipc_perm *in, struct ipc64_perm *out);
 void ipc64_perm_to_ipc_perm(struct ipc64_perm *in, struct ipc_perm *out);
+void ipc_update_perm(struct ipc64_perm *in, struct kern_ipc_perm *out);

 #if defined(__ia64__) || defined(__x86_64__) || defined(__hppa__) || defined(__XTENSA__)
   /* On IA-64, we always use the "64-bit version" of the IPC structures.  */
```

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 2.6.24-rc8-mm1 08/15] IPC: consolidate all xxxctl_down() functions

Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:37 GMT

View Forum Message <> Reply to Message

semctl_down(), msgctl_down() and shmctl_down() are used to handle the same
set of commands for each kind of IPC. They all start to do the same job (they
retrieve the ipc and do some permission checks) before handling the commands
on their own.

This patch proposes to consolidate this by moving these same pieces of code
into one common function called ipcctl_pre_down().
It simplifies a little these xxxctl_down() functions and increases a little
the maintainability.

Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
Acked-by: Serge Hallyn <serue@us.ibm.com>

```
 ipc/msg.c  |   48 +++++-------------------------------------------
 ipc/sem.c  |   42 ++++-------------------------------------
 ipc/shm.c  |   42 ++++-------------------------------------
 ipc/util.c |   51 ++++++++++++++++++++++++++++++++++++++++++++++++++
 ipc/util.h |    2 ++
 5 files changed, 66 insertions(+), 119 deletions(-)

Index: b/ipc/sem.c
===================================================================
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -142,21 +142,6 @@ void __init sem_init (void)
 }

 /*
- * This routine is called in the paths where the rw_mutex is held to protect
- * access to the idr tree.
- */
-static inline struct sem_array *sem_lock_check_down(struct ipc_namespace *ns,
-      int id)
-{
- struct kern_ipc_perm *ipcp = ipc_lock_check_down(&sem_ids(ns), id);
-
- if (IS_ERR(ipcp))
-  return (struct sem_array *)ipcp;
-
- return container_of(ipcp, struct sem_array, sem_perm);
-}
-
-/*
  * sem_lock_(check_) routines are called in the paths where the rw_mutex
  * is not held.
  */
@@ -880,31 +865,12 @@ static int semctl_down(struct ipc_namesp
   if (copy_semid_from_user(&semid64, arg.buf, version))
    return -EFAULT;
  }
- down_write(&sem_ids(ns).rw_mutex);
- sma = sem_lock_check_down(ns, semid);
- if (IS_ERR(sma)) {
-  err = PTR_ERR(sma);
-  goto out_up;
- }
-
- ipcp = &sma->sem_perm;

- err = audit_ipc_obj(ipcp);
- if (err)
-  goto out_unlock;
+ ipcp = ipcctl_pre_down(&sem_ids(ns), semid, cmd, &semid64.sem_perm, 0);
+ if (IS_ERR(ipcp))
+  return PTR_ERR(ipcp);

- if (cmd == IPC_SET) {
-  err = audit_ipc_set_perm(0, semid64.sem_perm.uid,
-      semid64.sem_perm.gid,
-      semid64.sem_perm.mode);
-  if (err)
-   goto out_unlock;
- }
- if (current->euid != ipcp->cuid &&
-     current->euid != ipcp->uid && !capable(CAP_SYS_ADMIN)) {
-      err=-EPERM;
-  goto out_unlock;
- }
+ sma = container_of(ipcp, struct sem_array, sem_perm);

  err = security_sem_semctl(sma, cmd);
  if (err)
Index: b/ipc/util.c
===================================================================
--- a/ipc/util.c
+++ b/ipc/util.c
@@ -774,6 +774,57 @@ void ipc_update_perm(struct ipc64_perm *
   | (in->mode & S_IRWXUGO);
 }

+/**
+ * ipcctl_pre_down - retrieve an ipc and check permissions for some IPC_XXX cmd
+ * @ids:  the table of ids where to look for the ipc
+ * @id:   the id of the ipc to retrieve
+ * @cmd:  the cmd to check
+ * @perm: the permission to set
+ * @extra_perm: one extra permission parameter used by msq
+ *
+ * This function does some common audit and permissions check for some IPC_XXX
+ * cmd and is called from semctl_down, shmctl_down and msgctl_down.
+ * It must be called without any lock held and
+ *  - retrieves the ipc with the given id in the given table.
+ *  - performs some audit and permission check, depending on the given cmd
+ *  - returns the ipc with both ipc and rw_mutex locks held in case of success
+ *    or an err-code without any lock held otherwise.
+ */
+struct kern_ipc_perm *ipcctl_pre_down(struct ipc_ids *ids, int id, int cmd,
+          struct ipc64_perm *perm, int extrat_perm)
+{
```

```
+ struct kern_ipc_perm *ipcp;
+ int err;
+
+ down_write(&ids->rw_mutex);
+ ipcp = ipc_lock_check_down(ids, id);
+ if (IS_ERR(ipcp)) {
+  err = PTR_ERR(ipcp);
+  goto out_up;
+ }
+
+ err = audit_ipc_obj(ipcp);
+ if (err)
+  goto out_unlock;
+
+ if (cmd == IPC_SET) {
+  err = audit_ipc_set_perm(extrat_perm, perm->uid,
+      perm->gid, perm->mode);
+  if (err)
+   goto out_unlock;
+ }
+ if (current->euid == ipcp->cuid ||
+     current->euid == ipcp->uid || capable(CAP_SYS_ADMIN))
+  return ipcp;
+
+ err = -EPERM;
+out_unlock:
+ ipc_unlock(ipcp);
+out_up:
+ up_write(&ids->rw_mutex);
+ return ERR_PTR(err);
+}
+
 #ifdef __ARCH_WANT_IPC_PARSE_VERSION


Index: b/ipc/util.h
===================================================================
--- a/ipc/util.h
+++ b/ipc/util.h
@@ -113,6 +113,8 @@ struct kern_ipc_perm *ipc_lock(struct ip
 void kernel_to_ipc64_perm(struct kern_ipc_perm *in, struct ipc64_perm *out);
 void ipc64_perm_to_ipc_perm(struct ipc64_perm *in, struct ipc_perm *out);
 void ipc_update_perm(struct ipc64_perm *in, struct kern_ipc_perm *out);
+struct kern_ipc_perm *ipcctl_pre_down(struct ipc_ids *ids, int id, int cmd,
+          struct ipc64_perm *perm, int extrat_perm);

 #if defined(__ia64__) || defined(__x86_64__) || defined(__hppa__) || defined(__XTENSA__)
   /* On IA-64, we always use the "64-bit version" of the IPC structures.  */
```

```
Index: b/ipc/msg.c
===================================================================
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -104,21 +104,6 @@ void __init msg_init(void)
 }

 /*
- * This routine is called in the paths where the rw_mutex is held to protect
- * access to the idr tree.
- */
-static inline struct msg_queue *msg_lock_check_down(struct ipc_namespace *ns,
-      int id)
-{
- struct kern_ipc_perm *ipcp = ipc_lock_check_down(&msg_ids(ns), id);
-
- if (IS_ERR(ipcp))
-  return (struct msg_queue *)ipcp;
-
- return container_of(ipcp, struct msg_queue, q_perm);
-}
-
-/*
  * msg_lock_(check_) routines are called in the paths where the rw_mutex
  * is not held.
  */
@@ -400,35 +385,12 @@ static int msgctl_down(struct ipc_namesp
    return -EFAULT;
  }

- down_write(&msg_ids(ns).rw_mutex);
- msq = msg_lock_check_down(ns, msqid);
- if (IS_ERR(msq)) {
-  err = PTR_ERR(msq);
-  goto out_up;
- }
-
- ipcp = &msq->q_perm;
-
- err = audit_ipc_obj(ipcp);
- if (err)
-  goto out_unlock;
-
- if (cmd == IPC_SET) {
-  err = audit_ipc_set_perm(msqid64.msg_qbytes,
-      msqid64.msg_perm.uid,
-      msqid64.msg_perm.gid,
-      msqid64.msg_perm.mode);
-  if (err)
-   goto out_unlock;
- }
+ ipcp = ipcctl_pre_down(&msg_ids(ns), msqid, cmd,
+          &msqid64.msg_perm, msqid64.msg_qbytes);
+ if (IS_ERR(ipcp))
+  return PTR_ERR(ipcp);

- if (current->euid != ipcp->cuid &&
-     current->euid != ipcp->uid &&
-     !capable(CAP_SYS_ADMIN)) {
-  /* We _could_ check for CAP_CHOWN above, but we don't */
-  err = -EPERM;
-  goto out_unlock;
- }
+ msq = container_of(ipcp, struct msg_queue, q_perm);

  err = security_msg_queue_msgctl(msq, cmd);
  if (err)
Index: b/ipc/shm.c
===================================================================
--- a/ipc/shm.c
+++ b/ipc/shm.c
@@ -127,18 +127,6 @@ static inline struct shmid_kernel *shm_l
  return container_of(ipcp, struct shmid_kernel, shm_perm);
 }

-static inline struct shmid_kernel *shm_lock_check_down(
-      struct ipc_namespace *ns,
-      int id)
-{
- struct kern_ipc_perm *ipcp = ipc_lock_check_down(&shm_ids(ns), id);
-
- if (IS_ERR(ipcp))
-  return (struct shmid_kernel *)ipcp;
-
- return container_of(ipcp, struct shmid_kernel, shm_perm);
-}
-
 /*
  * shm_lock_(check_) routines are called in the paths where the rw_mutex
  * is not held.
@@ -629,33 +617,11 @@ static int shmctl_down(struct ipc_namesp
    return -EFAULT;
  }

- down_write(&shm_ids(ns).rw_mutex);
- shp = shm_lock_check_down(ns, shmid);
- if (IS_ERR(shp)) {
-  err = PTR_ERR(shp);
-  goto out_up;
- }
-
- ipcp = &shp->shm_perm;
-
- err = audit_ipc_obj(ipcp);
- if (err)
-  goto out_unlock;
-
- if (cmd == IPC_SET) {
-  err = audit_ipc_set_perm(0, shmid64.shm_perm.uid,
-      shmid64.shm_perm.gid,
-      shmid64.shm_perm.mode);
-  if (err)
-   goto out_unlock;
- }
+ ipcp = ipcctl_pre_down(&shm_ids(ns), shmid, cmd, &shmid64.shm_perm, 0);
+ if (IS_ERR(ipcp))
+  return PTR_ERR(ipcp);

- if (current->euid != ipcp->uid &&
-     current->euid != ipcp->cuid &&
-     !capable(CAP_SYS_ADMIN)) {
-  err = -EPERM;
-  goto out_unlock;
- }
+ shp = container_of(ipcp, struct shmid_kernel, shm_perm);

  err = security_shm_shmctl(shp, cmd);
  if (err)
```

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID
Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:38 GMT
View Forum Message <> Reply to Message

From: Pierre Peiffer <pierre.peiffer@bull.net>

This patch provides three new API to change the ID of an existing
System V IPCs.

These APIs are:

```
 long msg_chid(struct ipc_namespace *ns, int id, int newid);
 long sem_chid(struct ipc_namespace *ns, int id, int newid);
 long shm_chid(struct ipc_namespace *ns, int id, int newid);
```

They return 0 or an error code in case of failure.

They may be useful for setting a specific ID for an IPC when preparing
a restart operation.

To be successful, the following rules must be respected:
- the IPC exists (of course...)
- the new ID must satisfy the ID computation rule.
- the entry in the idr corresponding to the new ID must be free.

Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
Acked-by: Serge Hallyn <serue@us.ibm.com>

```
 include/linux/msg.h |    2 ++
 include/linux/sem.h |    2 ++
 include/linux/shm.h |    3 +++
 ipc/msg.c           |   45 +++++++++++++++++++++++++++++++++++++++++++++
 ipc/sem.c           |   51 +++++++++++++++++++++++++++++++++++++++++++++++++++
 ipc/shm.c           |   45 +++++++++++++++++++++++++++++++++++++++++++++
 ipc/util.c          |   48 ++++++++++++++++++++++++++++++++++++++++++++++++
 ipc/util.h          |    1 +
 8 files changed, 197 insertions(+)

Index: b/include/linux/msg.h
===================================================================
--- a/include/linux/msg.h
+++ b/include/linux/msg.h
@@ -63,6 +63,7 @@ struct msginfo {

 #ifdef __KERNEL__
 #include <linux/list.h>
+#include <linux/ipc_namespace.h>

 /* one msg_msg structure for each message */
 struct msg_msg {
@@ -96,6 +97,7 @@ extern long do_msgsnd(int msqid, long mt
    size_t msgsz, int msgflg);
 extern long do_msgrcv(int msqid, long *pmtype, void __user *mtext,
    size_t msgsz, long msgtyp, int msgflg);
```

```
+long msg_chid(struct ipc_namespace *ns, int id, int newid);
 
 #endif /* __KERNEL__ */
 
Index: b/include/linux/sem.h
===================================================================
--- a/include/linux/sem.h
+++ b/include/linux/sem.h
@@ -138,9 +138,11 @@ struct sysv_sem {
 };
 
 #ifdef CONFIG_SYSVIPC
+#include <linux/ipc_namespace.h>
 
 extern int copy_semundo(unsigned long clone_flags, struct task_struct *tsk);
 extern void exit_sem(struct task_struct *tsk);
+long sem_chid(struct ipc_namespace *ns, int id, int newid);
 
 #else
 static inline int copy_semundo(unsigned long clone_flags, struct task_struct *tsk)
Index: b/include/linux/shm.h
===================================================================
--- a/include/linux/shm.h
+++ b/include/linux/shm.h
@@ -104,8 +104,11 @@ struct shmid_kernel /* private to the ke
 #define SHM_NORESERVE   010000  /* don't check for reservations */
 
 #ifdef CONFIG_SYSVIPC
+#include <linux/ipc_namespace.h>
+
 long do_shmat(int shmid, char __user *shmaddr, int shmflg, unsigned long *addr);
 extern int is_file_shm_hugepages(struct file *file);
+long shm_chid(struct ipc_namespace *ns, int id, int newid);
 #else
 static inline long do_shmat(int shmid, char __user *shmaddr,
     int shmflg, unsigned long *addr)
Index: b/ipc/msg.c
===================================================================
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -291,6 +291,51 @@ asmlinkage long sys_msgget(key_t key, in
  return ipcget(ns, &msg_ids(ns), &msg_ops, &msg_params);
 }
 
+/* must be called with mutex and msq locks held */
+static long msg_chid_nolock(struct ipc_namespace *ns, struct msg_queue *msq,
+       int newid)
+{
```

```
+ long err;
+
+ err = ipc_chid(&msg_ids(ns), msq->q_perm.id, newid);
+ if (!err)
+  msq->q_ctime = get_seconds();
+
+ return err;
+}
+
+/* API to use for changing an id from kernel space, not from the syscall, as
+   there is no permission check done here */
+long msg_chid(struct ipc_namespace *ns, int id, int newid)
+{
+ long err;
+ struct msg_queue *msq;
+
+retry:
+ err = idr_pre_get(&msg_ids(ns).ipcs_idr, GFP_KERNEL);
+ if (!err)
+  return -ENOMEM;
+
+ down_write(&msg_ids(ns).rw_mutex);
+ msq = msg_lock_check(ns, id);
+
+ if (IS_ERR(msq)) {
+  up_write(&msg_ids(ns).rw_mutex);
+  return PTR_ERR(msq);
+ }
+
+ err = msg_chid_nolock(ns, msq, newid);
+
+ msg_unlock(msq);
+ up_write(&msg_ids(ns).rw_mutex);
+
+ /* ipc_chid may return -EAGAIN in case of memory requirement */
+ if (err == -EAGAIN)
+  goto retry;
+
+ return err;
+}
+
 static inline unsigned long
 copy_msqid_to_user(void __user *buf, struct msqid64_ds *in, int version)
 {
Index: b/ipc/sem.c
===================================================================
--- a/ipc/sem.c
+++ b/ipc/sem.c
```

```
@@ -564,6 +564,57 @@ static void freeary(struct ipc_namespace
  ipc_rcu_putref(sma);
 }
 
+/* must be called with rw_mutex and sma locks held */
+static long sem_chid_nolock(struct ipc_namespace *ns, struct sem_array *sma,
+       int newid)
+{
+ long err;
+
+ err = ipc_chid(&sem_ids(ns), sma->sem_perm.id, newid);
+
+ if (!err) {
+  struct sem_undo *un;
+  for (un = sma->undo; un; un = un->id_next)
+   un->semid = newid;
+
+  sma->sem_ctime = get_seconds();
+ }
+
+ return err;
+}
+
+/* API to use for changing an id from kernel space, not from the syscall, as
+   there is no permission check done here */
+long sem_chid(struct ipc_namespace *ns, int id, int newid)
+{
+ long err;
+ struct sem_array *sma;
+
+retry:
+ err = idr_pre_get(&sem_ids(ns).ipcs_idr, GFP_KERNEL);
+ if (!err)
+  return -ENOMEM;
+
+ down_write(&sem_ids(ns).rw_mutex);
+ sma = sem_lock_check(ns, id);
+
+ if (IS_ERR(sma)) {
+  up_write(&sem_ids(ns).rw_mutex);
+  return PTR_ERR(sma);
+ }
+
+ err = sem_chid_nolock(ns, sma, newid);
+
+ sem_unlock(sma);
+ up_write(&sem_ids(ns).rw_mutex);
+
+ /* ipc_chid may return -EAGAIN in case of memory requirement */
+ if (err == -EAGAIN)
+  goto retry;
+
+ return err;
+}
+
 static unsigned long copy_semid_to_user(void __user *buf, struct semid64_ds *in, int version)
 {
  switch(version) {
Index: b/ipc/shm.c
===================================================================
--- a/ipc/shm.c
+++ b/ipc/shm.c
@@ -162,7 +162,52 @@ static inline int shm_addid(struct ipc_n
  return ipc_addid(&shm_ids(ns), &shp->shm_perm, ns->shm_ctlmni);
 }

+/* must be called with mutex and shp locks held */
+static long shm_chid_nolock(struct ipc_namespace *ns, struct shmid_kernel *shp,
+       int newid)
+{
+ long err;
+
+ err = ipc_chid(&shm_ids(ns), shp->shm_perm.id, newid);
+ if (!err) {
+  shp->shm_file->f_dentry->d_inode->i_ino = newid;
+  shp->shm_ctim = get_seconds();
+ }
+
+ return err;
+}
+
+/* API to use for changing an id from kernel space, not from the syscall, as
+   there is no permission check done here */
+long shm_chid(struct ipc_namespace *ns, int id, int newid)
+{
+ long err;
+ struct shmid_kernel *shp;
+
+retry:
+ err = idr_pre_get(&shm_ids(ns).ipcs_idr, GFP_KERNEL);
+ if (!err)
+  return -ENOMEM;
+
+ down_write(&shm_ids(ns).rw_mutex);
+ shp = shm_lock_check(ns, id);
+
+ if (IS_ERR(shp)) {
+  up_write(&shm_ids(ns).rw_mutex);
+  return PTR_ERR(shp);
+ }
+
+ err = shm_chid_nolock(ns, shp, newid);

+ shm_unlock(shp);
+ up_write(&shm_ids(ns).rw_mutex);
+
+ /* ipc_chid may return -EAGAIN in case of memory requirement */
+ if (err == -EAGAIN)
+   goto retry;
+
+ return err;
+}

 /* This is called by fork, once for every shm attach. */
 static void shm_open(struct vm_area_struct *vma)
Index: b/ipc/util.c
===================================================================
--- a/ipc/util.c
+++ b/ipc/util.c
@@ -363,6 +363,54 @@ retry:


 /**
+ * ipc_chid  - change an IPC identifier
+ * @ids: IPC identifier set
+ * @oldid: ID of the IPC permission set to move
+ * @newid: new ID of the IPC permission set to move
+ *
+ * Move an entry in the IPC idr from the 'oldid' place to the
+ *      'newid' place. The seq number of the entry is updated to match the
+ *      'newid' value.
+ *
+ * Called with the ipc lock and ipc_ids.rw_mutex held.
+ */
+int ipc_chid(struct ipc_ids *ids, int oldid, int newid)
+{
+ struct kern_ipc_perm *p;
+ int old_lid = oldid % SEQ_MULTIPLIER;
+ int new_lid = newid % SEQ_MULTIPLIER;
+
+ if (newid !=  (new_lid + (newid/SEQ_MULTIPLIER)*SEQ_MULTIPLIER))
+  return -EINVAL;
+
+ p = idr_find(&ids->ipcs_idr, old_lid);
```

```
+
+ if (!p)
+  return -EINVAL;
+
+ /* The idx in the idr may be the same but not the seq number. */
+ if (new_lid != old_lid) {
+  int id, err;
+
+  err = idr_get_new_above(&ids->ipcs_idr, p, new_lid, &id);
+  if (err)
+   return err;
+
+  /* do we get our wished id ? */
+  if (id == new_lid) {
+   idr_remove(&ids->ipcs_idr, old_lid);
+  } else {
+   idr_remove(&ids->ipcs_idr, id);
+   return -EBUSY;
+  }
+ }
+
+ p->id = newid;
+ p->seq = newid/SEQ_MULTIPLIER;
+ return 0;
+}
+
+/**
  * ipc_rmid - remove an IPC identifier
  * @ids: IPC identifier set
  * @ipcp: ipc perm structure containing the identifier to remove
Index: b/ipc/util.h
===================================================================
--- a/ipc/util.h
+++ b/ipc/util.h
@@ -85,6 +85,7 @@ int ipc_get_maxid(struct ipc_ids *);
 void ipc_rmid(struct ipc_ids *, struct kern_ipc_perm *);

 /* must be called with ipcp locked */
+int ipc_chid(struct ipc_ids *ids, int oldid, int newid);
 int ipcperms(struct kern_ipc_perm *ipcp, short flg);

 /* for rare, potentially huge allocations.
```

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org

https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 2.6.24-rc8-mm1 10/15] (RFC) IPC: new IPC_SETID command to modify an ID

Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:39 GMT

View Forum Message <> Reply to Message

From: Pierre Peiffer <pierre.peiffer@bull.net>

This patch adds a new IPC_SETID command to the System V IPCs set of commands,
which allows to change the ID of an existing IPC.

This command can be used through the semctl/shmctl/msgctl API, with the new
ID passed as the third argument for msgctl and shmctl (instead of a pointer)
and through the fourth argument for semctl.

To be successful, the following rules must be respected:
- the IPC exists
- the user must be allowed to change the IPC attributes regarding the IPC
  permissions.
- the new ID must satisfy the ID computation rule.
- the entry (in the kernel internal table of IPCs) corresponding to the new
  ID must be free.

Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
Acked-by: Serge Hallyn <serue@us.ibm.com>

```
 include/linux/ipc.h      |    9 +++++----
 ipc/compat.c             |    3 +++
 ipc/msg.c                |   27 ++++++++++++++++++++++++++-
 ipc/sem.c                |   27 ++++++++++++++++++++++++++-
 ipc/shm.c                |   27 ++++++++++++++++++++++++++-
 security/selinux/hooks.c |    3 +++
 6 files changed, 89 insertions(+), 7 deletions(-)

Index: b/include/linux/ipc.h
===================================================================
--- a/include/linux/ipc.h
+++ b/include/linux/ipc.h
@@ -35,10 +35,11 @@ struct ipc_perm
  * Control commands used with semctl, msgctl and shmctl
  * see also specific commands in sem.h, msg.h and shm.h
  */
-#define IPC_RMID 0     /* remove resource */
-#define IPC_SET  1     /* set ipc_perm options */
-#define IPC_STAT 2     /* get ipc_perm options */
```

```
-#define IPC_INFO 3     /* see ipcs */
+#define IPC_RMID   0     /* remove resource */
+#define IPC_SET    1     /* set ipc_perm options */
+#define IPC_STAT   2     /* get ipc_perm options */
+#define IPC_INFO   3     /* see ipcs */
+#define IPC_SETID  4     /* set ipc ID */

 /*
  * Version flags for semctl, msgctl, and shmctl commands
Index: b/ipc/compat.c
===================================================================
--- a/ipc/compat.c
+++ b/ipc/compat.c
@@ -253,6 +253,7 @@ long compat_sys_semctl(int first, int se
  switch (third & (~IPC_64)) {
  case IPC_INFO:
  case IPC_RMID:
+ case IPC_SETID:
  case SEM_INFO:
  case GETVAL:
  case GETPID:
@@ -425,6 +426,7 @@ long compat_sys_msgctl(int first, int se
  switch (second & (~IPC_64)) {
  case IPC_INFO:
  case IPC_RMID:
+ case IPC_SETID:
  case MSG_INFO:
   err = sys_msgctl(first, second, uptr);
   break;
@@ -597,6 +599,7 @@ long compat_sys_shmctl(int first, int se

  switch (second & (~IPC_64)) {
  case IPC_RMID:
+ case IPC_SETID:
  case SHM_LOCK:
  case SHM_UNLOCK:
   err = sys_shmctl(first, second, uptr);
Index: b/ipc/msg.c
===================================================================
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -329,7 +329,8 @@ retry:
  msg_unlock(msq);
  up_write(&msg_ids(ns).rw_mutex);

- /* ipc_chid may return -EAGAIN in case of memory requirement */
+ /* msg_chid_nolock may return -EAGAIN if there is no more free idr
+    entry, just go and retry by filling again de idr cache */
```

```
  if (err == -EAGAIN)
   goto retry;

@@ -465,6 +466,9 @@ static int msgctl_down(struct ipc_namesp
    */
   ss_wakeup(&msq->q_senders, 0);
   break;
+ case IPC_SETID:
+  err = msg_chid_nolock(ns, msq, (int)(long)buf);
+  break;
  default:
   err = -EINVAL;
  }
@@ -475,6 +479,24 @@ out_up:
  return err;
 }

+static int msgctl_setid(struct ipc_namespace *ns, int msqid, int cmd,
+   struct msqid_ds __user *buf, int version)
+{
+ int err;
+retry:
+ err = idr_pre_get(&msg_ids(ns).ipcs_idr, GFP_KERNEL);
+ if (!err)
+  return -ENOMEM;
+
+ err = msgctl_down(ns, msqid, cmd, buf, version);
+
+ /* msgctl_down may return -EAGAIN if there is no more free idr
+    entry, just go and retry by filling again de idr cache */
+ if (err == -EAGAIN)
+  goto retry;
+ return err;
+}
+
 asmlinkage long sys_msgctl(int msqid, int cmd, struct msqid_ds __user *buf)
 {
  struct msg_queue *msq;
@@ -575,6 +597,9 @@ asmlinkage long sys_msgctl(int msqid, in
  case IPC_RMID:
   err = msgctl_down(ns, msqid, cmd, buf, version);
   return err;
+ case IPC_SETID:
+  err = msgctl_setid(ns, msqid, cmd, buf, version);
+  return err;
  default:
   return  -EINVAL;
  }
```

```
Index: b/ipc/sem.c
===================================================================
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -608,7 +608,8 @@ retry:
  sem_unlock(sma);
  up_write(&sem_ids(ns).rw_mutex);

- /* ipc_chid may return -EAGAIN in case of memory requirement */
+ /* sem_chid_nolock may return -EAGAIN if there is no more free idr
+    entry, just go and retry by filling again de idr cache */
  if (err == -EAGAIN)
   goto retry;

@@ -935,6 +936,9 @@ static int semctl_down(struct ipc_namesp
   ipc_update_perm(&semid64.sem_perm, ipcp);
   sma->sem_ctime = get_seconds();
   break;
+ case IPC_SETID:
+  err = sem_chid_nolock(ns, sma, (int)arg.val);
+  break;
  default:
   err = -EINVAL;
  }
@@ -946,6 +950,24 @@ out_up:
  return err;
 }

+static int semctl_setid(struct ipc_namespace *ns,  int semid,
+   int cmd, int version, union semun arg)
+{
+ int err;
+retry:
+ err = idr_pre_get(&sem_ids(ns).ipcs_idr, GFP_KERNEL);
+ if (!err)
+  return -ENOMEM;
+
+ err = semctl_down(ns, semid, cmd, version, arg);
+
+ /* semctl_down may return -EAGAIN if there is no more free idr
+    entry, just go and retry by filling again de idr cache */
+ if (err == -EAGAIN)
+  goto retry;
+ return err;
+}
+
 asmlinkage long sys_semctl (int semid, int semnum, int cmd, union semun arg)
 {
  int err = -EINVAL;
@@ -978,6 +1000,9 @@ asmlinkage long sys_semctl (int semid, i
  case IPC_SET:
   err = semctl_down(ns, semid, cmd, version, arg);
   return err;
+ case IPC_SETID:
+  err = semctl_setid(ns, semid, cmd, version, arg);
+  return err;
  default:
   return -EINVAL;
  }
Index: b/ipc/shm.c
===================================================================
--- a/ipc/shm.c
+++ b/ipc/shm.c
@@ -202,7 +202,8 @@ retry:
  shm_unlock(shp);
  up_write(&shm_ids(ns).rw_mutex);

- /* ipc_chid may return -EAGAIN in case of memory requirement */
+ /* shm_chid_nolock may return -EAGAIN if there is no more free idr
+    entry, just go and retry by filling again de idr cache */
  if (err == -EAGAIN)
    goto retry;

@@ -679,6 +680,9 @@ static int shmctl_down(struct ipc_namesp
   ipc_update_perm(&shmid64.shm_perm, ipcp);
   shp->shm_ctim = get_seconds();
   break;
+ case IPC_SETID:
+  err = shm_chid_nolock(ns, shp, (int)(long)buf);
+  break;
  default:
   err = -EINVAL;
  }
@@ -689,6 +693,24 @@ out_up:
  return err;
 }

+static int shmctl_setid(struct ipc_namespace *ns, int shmid, int cmd,
+   struct shmid_ds __user *buf, int version)
+{
+ int err;
+retry:
+ err = idr_pre_get(&shm_ids(ns).ipcs_idr, GFP_KERNEL);
+ if (!err)
+  return -ENOMEM;
+
+ err = shmctl_down(ns, shmid, cmd, buf, version);
+
+ /* shmctl_down may return -EAGAIN if there is no more free idr
+    entry, just go and retry by filling again de idr cache */
+ if (err == -EAGAIN)
+  goto retry;
+ return err;
+}
+
 asmlinkage long sys_shmctl(int shmid, int cmd, struct shmid_ds __user *buf)
 {
  struct shmid_kernel *shp;
@@ -850,6 +872,9 @@ asmlinkage long sys_shmctl(int shmid, in
  case IPC_SET:
   err = shmctl_down(ns, shmid, cmd, buf, version);
   return err;
+ case IPC_SETID:
+  err = shmctl_setid(ns, shmid, cmd, buf, version);
+  return err;
  default:
   return -EINVAL;
  }
Index: b/security/selinux/hooks.c
===================================================================
--- a/security/selinux/hooks.c
+++ b/security/selinux/hooks.c
@@ -4651,6 +4651,7 @@ static int selinux_msg_queue_msgctl(stru
   perms = MSGQ__GETATTR | MSGQ__ASSOCIATE;
   break;
  case IPC_SET:
+ case IPC_SETID:
   perms = MSGQ__SETATTR;
   break;
  case IPC_RMID:
@@ -4799,6 +4800,7 @@ static int selinux_shm_shmctl(struct shm
   perms = SHM__GETATTR | SHM__ASSOCIATE;
   break;
  case IPC_SET:
+ case IPC_SETID:
   perms = SHM__SETATTR;
   break;
  case SHM_LOCK:
@@ -4910,6 +4912,7 @@ static int selinux_sem_semctl(struct sem
   perms = SEM__DESTROY;
   break;
  case IPC_SET:
+ case IPC_SETID:
   perms = SEM__SETATTR;
   break;
  case IPC_STAT:
```

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 2.6.24-rc8-mm1 11/15] (RFC) IPC: new IPC_SETALL command to modify all settings

Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:40 GMT

View Forum Message <> Reply to Message

From: Pierre Peiffer <pierre.peiffer@bull.net>

This patch adds a new IPC_SETALL command to the System V IPCs set of commands,
which allows to change all the settings of an IPC

It works exactly the same way as the IPC_SET command, except that it
additionally changes all the times and the pids values

Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
Acked-by: Serge Hallyn <serue@us.ibm.com>

```
 include/linux/ipc.h      |    1 +
 ipc/compat.c             |    3 +++
 ipc/msg.c                |   15 +++++++++++++--
 ipc/sem.c                |   10 +++++++++-
 ipc/shm.c                |   13 ++++++++++++-
 ipc/util.c               |    7 ++++++-
 security/selinux/hooks.c |    3 +++
 7 files changed, 47 insertions(+), 5 deletions(-)

Index: b/include/linux/ipc.h
===================================================================
--- a/include/linux/ipc.h
+++ b/include/linux/ipc.h
@@ -40,6 +40,7 @@ struct ipc_perm
 #define IPC_STAT   2     /* get ipc_perm options */
 #define IPC_INFO   3     /* see ipcs */
 #define IPC_SETID  4     /* set ipc ID */
+#define IPC_SETALL 5     /* set all parameters */

 /*
```

```
  * Version flags for semctl, msgctl, and shmctl commands
Index: b/ipc/compat.c
===================================================================
--- a/ipc/compat.c
+++ b/ipc/compat.c
@@ -282,6 +282,7 @@ long compat_sys_semctl(int first, int se
    err = -EFAULT;
   break;

+ case IPC_SETALL:
  case IPC_SET:
   if (version == IPC_64) {
    err = get_compat_semid64_ds(&s64, compat_ptr(pad));
@@ -431,6 +432,7 @@ long compat_sys_msgctl(int first, int se
   err = sys_msgctl(first, second, uptr);
   break;

+ case IPC_SETALL:
  case IPC_SET:
   if (version == IPC_64) {
    err = get_compat_msqid64(&m64, uptr);
@@ -621,6 +623,7 @@ long compat_sys_shmctl(int first, int se
   break;


+ case IPC_SETALL:
  case IPC_SET:
   if (version == IPC_64) {
    err = get_compat_shmid64_ds(&s64, uptr);
Index: b/ipc/msg.c
===================================================================
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -426,7 +426,7 @@ static int msgctl_down(struct ipc_namesp
  struct msg_queue *msq;
  int err;

- if (cmd == IPC_SET) {
+ if (cmd == IPC_SET || cmd == IPC_SETALL) {
   if (copy_msqid_from_user(&msqid64, buf, version))
    return -EFAULT;
  }
@@ -447,6 +447,7 @@ static int msgctl_down(struct ipc_namesp
   freeque(ns, ipcp);
   goto out_up;
  case IPC_SET:
+ case IPC_SETALL:
   if (msqid64.msg_qbytes > ns->msg_ctlmnb &&
       !capable(CAP_SYS_RESOURCE)) {
    err = -EPERM;
@@ -456,7 +457,14 @@ static int msgctl_down(struct ipc_namesp
   msq->q_qbytes = msqid64.msg_qbytes;

   ipc_update_perm(&msqid64.msg_perm, ipcp);
-  msq->q_ctime = get_seconds();
+  if (cmd == IPC_SETALL) {
+   msq->q_stime = msqid64.msg_stime;
+   msq->q_rtime = msqid64.msg_rtime;
+   msq->q_ctime = msqid64.msg_ctime;
+   msq->q_lspid = msqid64.msg_lspid;
+   msq->q_lrpid = msqid64.msg_lrpid;
+  } else
+   msq->q_ctime = get_seconds();
   /* sleeping receivers might be excluded by
    * stricter permissions.
    */
@@ -507,6 +515,8 @@ asmlinkage long sys_msgctl(int msqid, in
   return -EINVAL;

  version = ipc_parse_version(&cmd);
+ if (version < 0)
+  return -EINVAL;
  ns = current->nsproxy->ipc_ns;

  switch (cmd) {
@@ -594,6 +604,7 @@ asmlinkage long sys_msgctl(int msqid, in
   return success_return;
  }
  case IPC_SET:
+ case IPC_SETALL:
  case IPC_RMID:
   err = msgctl_down(ns, msqid, cmd, buf, version);
   return err;
Index: b/ipc/sem.c
===================================================================
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -913,7 +913,7 @@ static int semctl_down(struct ipc_namesp
  struct semid64_ds semid64;
  struct kern_ipc_perm *ipcp;

- if(cmd == IPC_SET) {
+ if (cmd == IPC_SET || cmd == IPC_SETALL) {
   if (copy_semid_from_user(&semid64, arg.buf, version))
    return -EFAULT;
  }
```

```
@@ -936,6 +936,11 @@ static int semctl_down(struct ipc_namesp
   ipc_update_perm(&semid64.sem_perm, ipcp);
   sma->sem_ctime = get_seconds();
   break;
+ case IPC_SETALL:
+  ipc_update_perm(&semid64.sem_perm, ipcp);
+  sma->sem_ctime = semid64.sem_ctime;
+  sma->sem_otime = semid64.sem_otime;
+  break;
  case IPC_SETID:
   err = sem_chid_nolock(ns, sma, (int)arg.val);
   break;
@@ -978,6 +983,8 @@ asmlinkage long sys_semctl (int semid, i
   return -EINVAL;

  version = ipc_parse_version(&cmd);
+ if (version < 0)
+  return -EINVAL;
  ns = current->nsproxy->ipc_ns;

  switch(cmd) {
@@ -998,6 +1005,7 @@ asmlinkage long sys_semctl (int semid, i
   return err;
  case IPC_RMID:
  case IPC_SET:
+ case IPC_SETALL:
   err = semctl_down(ns, semid, cmd, version, arg);
   return err;
  case IPC_SETID:
Index: b/ipc/shm.c
===================================================================
--- a/ipc/shm.c
+++ b/ipc/shm.c
@@ -658,7 +658,7 @@ static int shmctl_down(struct ipc_namesp
  struct shmid_kernel *shp;
  int err;

- if (cmd == IPC_SET) {
+ if (cmd == IPC_SET || cmd == IPC_SETALL) {
   if (copy_shmid_from_user(&shmid64, buf, version))
    return -EFAULT;
  }
@@ -680,6 +680,14 @@ static int shmctl_down(struct ipc_namesp
   ipc_update_perm(&shmid64.shm_perm, ipcp);
   shp->shm_ctim = get_seconds();
   break;
+ case IPC_SETALL:
+  ipc_update_perm(&shmid64.shm_perm, ipcp);
+  shp->shm_atim = shmid64.shm_atime;
+  shp->shm_dtim = shmid64.shm_dtime;
+  shp->shm_ctim = shmid64.shm_ctime;
+  shp->shm_cprid = shmid64.shm_cpid;
+  shp->shm_lprid = shmid64.shm_lpid;
+  break;
  case IPC_SETID:
   err = shm_chid_nolock(ns, shp, (int)(long)buf);
   break;
@@ -723,6 +731,8 @@ asmlinkage long sys_shmctl(int shmid, in
  }

  version = ipc_parse_version(&cmd);
+ if (version < 0)
+  return -EINVAL;
  ns = current->nsproxy->ipc_ns;

  switch (cmd) { /* replace with proc interface ? */
@@ -870,6 +880,7 @@ asmlinkage long sys_shmctl(int shmid, in
  }
  case IPC_RMID:
  case IPC_SET:
+ case IPC_SETALL:
   err = shmctl_down(ns, shmid, cmd, buf, version);
   return err;
  case IPC_SETID:
Index: b/ipc/util.c
===================================================================
--- a/ipc/util.c
+++ b/ipc/util.c
@@ -855,7 +855,7 @@ struct kern_ipc_perm *ipcctl_pre_down(st
  if (err)
   goto out_unlock;

- if (cmd == IPC_SET) {
+ if (cmd == IPC_SET || cmd == IPC_SETALL) {
   err = audit_ipc_set_perm(extrat_perm, perm->uid,
       perm->gid, perm->mode);
   if (err)
@@ -883,6 +883,8 @@ out_up:
  * Return IPC_64 for new style IPC and IPC_OLD for old style IPC.
  * The @cmd value is turned from an encoding command and version into
  * just the command code.
+ * In case of incompatibility between the command and the style, an
+ * errcode is returned.
  */

 int ipc_parse_version (int *cmd)
```

```
@@ -891,6 +893,9 @@ int ipc_parse_version (int *cmd)
   *cmd ^= IPC_64;
   return IPC_64;
  } else {
+  /* don't support this command for old ipc */
+  if (*cmd == IPC_SETALL)
+   return -EINVAL;
   return IPC_OLD;
  }
 }
Index: b/security/selinux/hooks.c
===================================================================
--- a/security/selinux/hooks.c
+++ b/security/selinux/hooks.c
@@ -4651,6 +4651,7 @@ static int selinux_msg_queue_msgctl(stru
   perms = MSGQ__GETATTR | MSGQ__ASSOCIATE;
   break;
  case IPC_SET:
+ case IPC_SETALL:
  case IPC_SETID:
   perms = MSGQ__SETATTR;
   break;
@@ -4800,6 +4801,7 @@ static int selinux_shm_shmctl(struct shm
   perms = SHM__GETATTR | SHM__ASSOCIATE;
   break;
  case IPC_SET:
+ case IPC_SETALL:
  case IPC_SETID:
   perms = SHM__SETATTR;
   break;
@@ -4912,6 +4914,7 @@ static int selinux_sem_semctl(struct sem
   perms = SEM__DESTROY;
   break;
  case IPC_SET:
+ case IPC_SETALL:
  case IPC_SETID:
   perms = SEM__SETATTR;
   break;
```

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 2.6.24-rc8-mm1 12/15] (RFC) IPC/semaphores: make use of RCU to free the sem_undo_list
Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:41 GMT

View Forum Message <> Reply to Message

From: Pierre Peiffer <pierre.peiffer@bull.net>

Today, the sem_undo_list is freed when the last task using it exits.
There is no mechanism in place, that allows a safe concurrent access to
the sem_undo_list of a target task and protects efficiently against a
task-exit.

That is okay for now as we don't need this.

As I would like to provide a /proc interface to access this data, I need
such a safe access, without blocking the target task if possible.

This patch proposes to introduce the use of RCU to delay the real free of
these sem_undo_list structures. They can then be accessed in a safe manner
by any tasks inside read critical section, this way:

```
 struct sem_undo_list *undo_list;
 int ret;
 ...
 rcu_read_lock();
 undo_list = rcu_dereference(task->sysvsem.undo_list);
 if (undo_list)
  ret = atomic_inc_not_zero(&undo_list->refcnt);
 rcu_read_unlock();
 ...
 if (undo_list && ret) {
  /* section where undo_list can be used quietly */
  ...
 }
 ...
```

Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>

```
 include/linux/sem.h |    7 +++++--
 ipc/sem.c           |   42 ++++++++++++++++++++++++++----------------
 2 files changed, 31 insertions(+), 18 deletions(-)

Index: b/include/linux/sem.h
===================================================================
--- a/include/linux/sem.h
+++ b/include/linux/sem.h
@@ -115,7 +115,8 @@ struct sem_queue {
 };
 /* Each task has a list of undo requests. They are executed automatically
- * when the process exits.
+ * when the last refcnt of sem_undo_list is released (ie when the process exits
+ * in the general case)
  */
 struct sem_undo {
  struct sem_undo * proc_next; /* next entry on this process */
@@ -125,12 +126,14 @@ struct sem_undo {
 };

 /* sem_undo_list controls shared access to the list of sem_undo structures
- * that may be shared among all a CLONE_SYSVSEM task group.
+ * that may be shared among all a CLONE_SYSVSEM task group or with an external
+ * process which changes the list through procfs.
  */
 struct sem_undo_list {
  atomic_t refcnt;
  spinlock_t lock;
  struct sem_undo *proc_list;
+ struct ipc_namespace *ns;
 };

 struct sysv_sem {
Index: b/ipc/sem.c
===================================================================
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -1038,6 +1038,7 @@ static inline int get_undo_list(struct s
    return -ENOMEM;
   spin_lock_init(&undo_list->lock);
   atomic_set(&undo_list->refcnt, 1);
+  undo_list->ns = get_ipc_ns(current->nsproxy->ipc_ns);
   current->sysvsem.undo_list = undo_list;
  }
  *undo_listp = undo_list;
@@ -1316,7 +1317,8 @@ int copy_semundo(unsigned long clone_fla
 }

 /*
- * add semadj values to semaphores, free undo structures.
+ * add semadj values to semaphores, free undo structures, if there is no
+ * more user.
  * undo structures are not freed when semaphore arrays are destroyed
  * so some of them may be out of date.
  * IMPLEMENTATION NOTE: There is some confusion over whether the
@@ -1326,23 +1328,17 @@ int copy_semundo(unsigned long clone_fla
  * The original implementation attempted to do this (queue and wait).
  * The current implementation does not do so. The POSIX standard
  * and SVID should be consulted to determine what behavior is mandated.
+ *
+ * Note:
+ * A concurrent task is only allowed to access and go through the list
+ * of sem_undo if it successfully grabs a refcnt.
  */
-void exit_sem(struct task_struct *tsk)
+static void free_semundo_list(struct sem_undo_list *undo_list)
 {
- struct sem_undo_list *undo_list;
  struct sem_undo *u, **up;
- struct ipc_namespace *ns;

- undo_list = tsk->sysvsem.undo_list;
- if (!undo_list)
-  return;
-
- 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.
+ /* There's no need to hold the semundo list lock, as there are
+  * no more tasks or possible users for this undo list.
   */
  for (up = &undo_list->proc_list; (u = *up); *up = u->proc_next, kfree(u)) {
   struct sem_array *sma;
@@ -1354,7 +1350,7 @@ void exit_sem(struct task_struct *tsk)

   if(semid == -1)
    continue;
-  sma = sem_lock(ns, semid);
+  sma = sem_lock(undo_list->ns, semid);
   if (IS_ERR(sma))
    continue;

@@ -1368,7 +1364,8 @@ void exit_sem(struct task_struct *tsk)
    if (u == un)
     goto found;
   }
-  printk ("exit_sem undo list error id=%d\n", u->semid);
+  printk(KERN_ERR "free_semundo_list error id=%d\n",
+         u->semid);
   goto next_entry;
 found:
   *unp = un->id_next;
@@ -1404,9 +1401,22 @@ found:
 next_entry:
   sem_unlock(sma);
  }
+ put_ipc_ns(undo_list->ns);
  kfree(undo_list);
 }

+/* called from do_exit() */
+void exit_sem(struct task_struct *tsk)
+{
+ struct sem_undo_list *ul = tsk->sysvsem.undo_list;
+ if (ul) {
+  rcu_assign_pointer(tsk->sysvsem.undo_list, NULL);
+  synchronize_rcu();
+  if (atomic_dec_and_test(&ul->refcnt))
+   free_semundo_list(ul);
+ }
+}
+
 #ifdef CONFIG_PROC_FS
 static int sysvipc_sem_proc_show(struct seq_file *s, void *it)
 {
```

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 2.6.24-rc8-mm1 13/15] (RFC) IPC/semaphores: per <pid> semundo file in procfs
Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:42 GMT

View Forum Message <> Reply to Message

From: Pierre Peiffer <pierre.peiffer@bull.net>

This patch adds a new procfs interface to display the per-process semundo data.

A new per-PID file is added, named "semundo".
It contains one line per semaphore IPC where there is something to undo for this process.
Then, each line contains the semid followed by each undo value corresponding to each semaphores of the semaphores array.

This interface will be specially useful to allow a user to access
these data, for example for checkpointing a process

Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
Acked-by: Serge Hallyn <serue@us.ibm.com>

```
 fs/proc/base.c     |    3 +
 fs/proc/internal.h |    1
 ipc/sem.c          |  153 +++++++++++++++++++++++++++++++++++++++++++++++++++++
 3 files changed, 157 insertions(+)

Index: b/fs/proc/base.c
===================================================================
--- a/fs/proc/base.c
+++ b/fs/proc/base.c
@@ -2255,6 +2255,9 @@ static const struct pid_entry tgid_base_
 #ifdef CONFIG_TASK_IO_ACCOUNTING
  INF("io", S_IRUGO, pid_io_accounting),
 #endif
+#ifdef CONFIG_SYSVIPC
+ REG("semundo",   S_IRUGO, semundo),
+#endif
 };

 static int proc_tgid_base_readdir(struct file * filp,
Index: b/fs/proc/internal.h
===================================================================
--- a/fs/proc/internal.h
+++ b/fs/proc/internal.h
@@ -64,6 +64,7 @@ extern const struct file_operations proc
 extern const struct file_operations proc_smaps_operations;
 extern const struct file_operations proc_clear_refs_operations;
 extern const struct file_operations proc_pagemap_operations;
+extern const struct file_operations proc_semundo_operations;

 void free_proc_entry(struct proc_dir_entry *de);

Index: b/ipc/sem.c
===================================================================
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -1435,4 +1435,157 @@ static int sysvipc_sem_proc_show(struct
      sma->sem_otime,
      sma->sem_ctime);
 }
+
+
+/* iterator */
+static void *semundo_start(struct seq_file *m, loff_t *ppos)
+{
+ struct sem_undo_list *undo_list = m->private;
+ struct sem_undo *undo;
+ loff_t pos = *ppos;
+
+ if (!undo_list)
+  return NULL;
+
+ if (pos < 0)
+  return NULL;
+
+ /* If undo_list is not NULL, it means that we've successfully grabbed
+  * a refcnt in semundo_open. That prevents the undo_list itself and the
+  * undo elements to be freed
+  */
+ spin_lock(&undo_list->lock);
+ undo = undo_list->proc_list;
+ while (undo) {
+  if ((undo->semid != -1) && !(pos--))
+   break;
+  undo = undo->proc_next;
+ }
+ spin_unlock(&undo_list->lock);
+
+ return undo;
+}
+
+static void *semundo_next(struct seq_file *m, void *v, loff_t *ppos)
+{
+ struct sem_undo *undo = v;
+ struct sem_undo_list *undo_list = m->private;
+
+ /*
+  * No need to protect against undo_list being NULL, if we are here,
+  * it can't be NULL.
+  * Moreover, by releasing the lock between each iteration, we allow the
+  * list to change between each iteration, but we only want to guarantee
+  * to have access to some valid data during the _show, not to have a
+  * full coherent view of the whole list.
+  */
+ spin_lock(&undo_list->lock);
+ do {
+  undo = undo->proc_next;
+ } while (undo && (undo->semid == -1));
+ ++*ppos;
+ spin_unlock(&undo_list->lock);
+
+ return undo;
+}
+
+static void semundo_stop(struct seq_file *m, void *v)
+{
+}
+
+static int semundo_show(struct seq_file *m, void *v)
+{
+ struct sem_undo_list *undo_list = m->private;
+ struct sem_undo *u = v;
+ int nsems, i;
+ struct sem_array *sma;
+
+ /*
+  * This semid has been deleted, ignore it.
+  * Even if we skipped all sem_undo belonging to deleted semid
+  * in semundo_next(), some more deletions may have happened.
+  */
+ if (u->semid == -1)
+  return 0;
+
+ seq_printf(m, "%10d", u->semid);
+
+ sma = sem_lock(undo_list->ns, u->semid);
+ if (IS_ERR(sma))
+  goto out;
+
+ nsems = sma->sem_nsems;
+ sem_unlock(sma);
+
+ for (i = 0; i < nsems; i++)
+  seq_printf(m, " %6d", u->semadj[i]);
+
+out:
+ seq_putc(m, '\n');
+ return 0;
+}
+
+static struct seq_operations semundo_op = {
+ .start = semundo_start,
+ .next = semundo_next,
+ .stop = semundo_stop,
+ .show = semundo_show
+};
+
+/*
```

```
+ * semundo_open: open operation for /proc/<PID>/semundo file
+ */
+static int semundo_open(struct inode *inode, struct file *file)
+{
+ struct task_struct *task;
+ struct sem_undo_list *undo_list = NULL;
+ int ret = 0;
+
+ /*
+  * We use RCU to be sure that the sem_undo_list will not be freed
+  * while we are accessing it. This may happen if the target task
+  * exits. Once we get a ref on it, we are ok.
+  */
+ rcu_read_lock();
+ task = get_pid_task(PROC_I(inode)->pid, PIDTYPE_PID);
+ if (task) {
+  undo_list = rcu_dereference(task->sysvsem.undo_list);
+  if (undo_list)
+   ret = !atomic_inc_not_zero(&undo_list->refcnt);
+  put_task_struct(task);
+ }
+ rcu_read_unlock();
+
+ if (!task || ret)
+  return -EINVAL;
+
+ ret = seq_open(file, &semundo_op);
+ if (!ret) {
+  struct seq_file *m = file->private_data;
+  m->private = undo_list;
+  return 0;
+ }
+
+ if (undo_list && atomic_dec_and_test(&undo_list->refcnt))
+  free_semundo_list(undo_list);
+ return ret;
+}
+
+static int semundo_release(struct inode *inode, struct file *file)
+{
+ struct seq_file *m = file->private_data;
+ struct sem_undo_list *undo_list = m->private;
+
+ if (undo_list && atomic_dec_and_test(&undo_list->refcnt))
+  free_semundo_list(undo_list);
+
+ return seq_release(inode, file);
+}
```

```
+
+const struct file_operations proc_semundo_operations = {
+ .open  = semundo_open,
+ .read  = seq_read,
+ .llseek  = seq_lseek,
+ .release = semundo_release,
+};
 #endif
```

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 2.6.24-rc8-mm1 14/15] (RFC) IPC/semaphores: prepare semundo code to work on another task than
Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:43 GMT

View Forum Message <> Reply to Message

From: Pierre Peiffer <pierre.peiffer@bull.net>

In order to modify the semundo-list of a task from procfs, we must be able to
work on any target task.
But all the existing code playing with the semundo-list, currently works
only on the 'current' task, and does not allow to specify any target task.

This patch changes all these routines to allow them to work on a specified
task, passed in parameter, instead of current.

This is mainly a preparation for the semundo_write() operation, on the
/proc/<pid>/semundo file, as provided in the next patch.

Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>

```
 ipc/sem.c |   90 ++++++++++++++++++++++++++++++++++++++++++++++++----------------
 1 file changed, 68 insertions(+), 22 deletions(-)

Index: b/ipc/sem.c
===================================================================
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -1017,8 +1017,9 @@ asmlinkage long sys_semctl (int semid, i
 }
 /* If the task doesn't already have a undo_list, then allocate one
- * here.  We guarantee there is only one thread using this undo list,
- * and current is THE ONE
+ * here.
+ * The target task (tsk) is current in the general case, except when
+ * accessed from the procfs (ie when writting to /proc/<pid>/semundo)
  *
  * If this allocation and assignment succeeds, but later
  * portions of this code fail, there is no need to free the sem_undo_list.
@@ -1026,22 +1027,60 @@ asmlinkage long sys_semctl (int semid, i
  * at exit time.
  *
  * This can block, so callers must hold no locks.
+ *
+ * Note: task_lock is used to synchronize 1. several possible concurrent
+ * creations and 2. the free of the undo_list (done when the task using it
+ * exits). In the second case, we check the PF_EXITING flag to not create
+ * an undo_list for a task which has exited.
+ * If there already is an undo_list for this task, there is no need
+ * to held the task-lock to retrieve it, as the pointer can not change
+ * afterwards.
  */
-static inline int get_undo_list(struct sem_undo_list **undo_listp)
+static inline int get_undo_list(struct task_struct *tsk,
+    struct sem_undo_list **ulp)
 {
- struct sem_undo_list *undo_list;
+ if (tsk->sysvsem.undo_list == NULL) {
+  struct sem_undo_list *undo_list;

- undo_list = current->sysvsem.undo_list;
- if (!undo_list) {
-  undo_list = kzalloc(sizeof(*undo_list), GFP_KERNEL);
+  /* we must alloc a new one */
+  undo_list = kmalloc(sizeof(*undo_list), GFP_KERNEL);
   if (undo_list == NULL)
    return -ENOMEM;
+
+  task_lock(tsk);
+
+  /* check again if there is an undo_list for this task */
+  if (tsk->sysvsem.undo_list) {
+   if (tsk != current)
+    atomic_inc(&tsk->sysvsem.undo_list->refcnt);
+   task_unlock(tsk);
+   kfree(undo_list);
+   goto out;
+  }
+
   spin_lock_init(&undo_list->lock);
-  atomic_set(&undo_list->refcnt, 1);
-  undo_list->ns = get_ipc_ns(current->nsproxy->ipc_ns);
-  current->sysvsem.undo_list = undo_list;
+  /*
+   * If tsk is not current (meaning that current is creating
+   * a semundo_list for a target task through procfs), and if
+   * it's not being exited then refcnt must be 2: the target
+   * task tsk + current.
+   */
+  if (tsk == current)
+   atomic_set(&undo_list->refcnt, 1);
+  else if (!(tsk->flags & PF_EXITING))
+   atomic_set(&undo_list->refcnt, 2);
+  else {
+   task_unlock(tsk);
+   kfree(undo_list);
+   return -EINVAL;
+  }
+  undo_list->ns = get_ipc_ns(tsk->nsproxy->ipc_ns);
+  undo_list->proc_list = NULL;
+  tsk->sysvsem.undo_list = undo_list;
+  task_unlock(tsk);
  }
- *undo_listp = undo_list;
+out:
+ *ulp = tsk->sysvsem.undo_list;
  return 0;
 }

@@ -1065,17 +1104,12 @@ static struct sem_undo *lookup_undo(stru
  return un;
 }

-static struct sem_undo *find_undo(struct ipc_namespace *ns, int semid)
+static struct sem_undo *find_undo(struct sem_undo_list *ulp, int semid)
 {
  struct sem_array *sma;
- struct sem_undo_list *ulp;
  struct sem_undo *un, *new;
+ struct ipc_namespace *ns;
  int nsems;
- int error;
-
- error = get_undo_list(&ulp);
- if (error)
-  return ERR_PTR(error);
```

```
  spin_lock(&ulp->lock);
  un = lookup_undo(ulp, semid);
@@ -1083,6 +1117,8 @@ static struct sem_undo *find_undo(struct
  if (likely(un!=NULL))
   goto out;

+ ns = ulp->ns;
+
  /* no undo structure around - allocate one. */
  sma = sem_lock_check(ns, semid);
  if (IS_ERR(sma))
@@ -1133,6 +1169,7 @@ asmlinkage long sys_semtimedop(int semid
  struct sem_array *sma;
  struct sembuf fast_sops[SEMOPM_FAST];
  struct sembuf* sops = fast_sops, *sop;
+ struct sem_undo_list *ulp;
  struct sem_undo *un;
  int undos = 0, alter = 0, max;
  struct sem_queue queue;
@@ -1177,9 +1214,13 @@ asmlinkage long sys_semtimedop(int semid
    alter = 1;
  }

+ error = get_undo_list(current, &ulp);
+ if (error)
+  goto out_free;
+
 retry_undos:
  if (undos) {
-  un = find_undo(ns, semid);
+  un = find_undo(ulp, semid);
   if (IS_ERR(un)) {
    error = PTR_ERR(un);
    goto out_free;
@@ -1305,7 +1346,7 @@ int copy_semundo(unsigned long clone_fla
  int error;

  if (clone_flags & CLONE_SYSVSEM) {
-  error = get_undo_list(&undo_list);
+  error = get_undo_list(current, &undo_list);
   if (error)
    return error;
   atomic_inc(&undo_list->refcnt);
@@ -1405,10 +1446,15 @@ next_entry:
  kfree(undo_list);
 }
```

```
-/* called from do_exit() */
+/* exit_sem: called from do_exit()
+ * task_lock is used to synchronize with get_undo_list()
+ */
 void exit_sem(struct task_struct *tsk)
 {
- struct sem_undo_list *ul = tsk->sysvsem.undo_list;
+ struct sem_undo_list *ul;
+ task_lock(tsk);
+ ul = tsk->sysvsem.undo_list;
+ task_unlock(tsk);
  if (ul) {
   rcu_assign_pointer(tsk->sysvsem.undo_list, NULL);
   synchronize_rcu();
```

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 2.6.24-rc8-mm1 15/15] (RFC) IPC/semaphores: add write() operation to semundo file in procfs
Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:44 GMT

View Forum Message <> Reply to Message

From: Pierre Peiffer <pierre.peiffer@bull.net>

This patch adds the write operation to the semundo file.
This write operation allows root to add or update the semundo list and
their values for a given process.

The user must provide some lines, each containing the semaphores ID
followed by the semaphores values to undo.

The operation failed if the given semaphore ID does not exist or if the
number of values does not match the number of semaphores in the array.

Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>

```
 fs/proc/base.c |    2
 ipc/sem.c      |  232 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++--
 2 files changed, 227 insertions(+), 7 deletions(-)
```

Index: b/fs/proc/base.c

```
===================================================================
--- a/fs/proc/base.c
+++ b/fs/proc/base.c
@@ -2256,7 +2256,7 @@ static const struct pid_entry tgid_base_
  INF("io", S_IRUGO, pid_io_accounting),
 #endif
 #ifdef CONFIG_SYSVIPC
- REG("semundo",   S_IRUGO, semundo),
+ REG("semundo",   S_IWUSR|S_IRUGO, semundo),
 #endif
 };

Index: b/ipc/sem.c
===================================================================
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -1580,6 +1580,9 @@ static struct seq_operations semundo_op

 /*
  * semundo_open: open operation for /proc/<PID>/semundo file
+ *
+ * If the file is opened in write mode and no semundo list exists for
+ * this target PID, it is created here.
  */
 static int semundo_open(struct inode *inode, struct file *file)
 {
@@ -1598,18 +1601,31 @@ static int semundo_open(struct inode *in
   undo_list = rcu_dereference(task->sysvsem.undo_list);
   if (undo_list)
    ret = !atomic_inc_not_zero(&undo_list->refcnt);
-  put_task_struct(task);
  }
  rcu_read_unlock();

- if (!task || ret)
+ if (!task)
   return -EINVAL;

- ret = seq_open(file, &semundo_op);
+ if (ret) {
+  put_task_struct(task);
+  return -EINVAL;
+ }
+
+
+ /* Create an undo_list if needed and if file is opened in write mode */
+ if (!undo_list && (file->f_flags & O_WRONLY || file->f_flags & O_RDWR))
+  ret = get_undo_list(task, &undo_list);
```

```
+
+ put_task_struct(task);
+
  if (!ret) {
-  struct seq_file *m = file->private_data;
-  m->private = undo_list;
-  return 0;
+  ret = seq_open(file, &semundo_op);
+  if (!ret) {
+   struct seq_file *m = file->private_data;
+   m->private = undo_list;
+   return 0;
+  }
  }

  if (undo_list && atomic_dec_and_test(&undo_list->refcnt))
@@ -1617,6 +1633,209 @@ static int semundo_open(struct inode *in
  return ret;
 }

+/* Skip all spaces at the beginning of the buffer */
+static inline int skip_space(const char __user **buf, size_t *len)
+{
+ char c = 0;
+ while (*len) {
+  if (get_user(c, *buf))
+   return -EFAULT;
+  if (c != '\t' && c != ' ')
+   break;
+  --*len;
+  ++*buf;
+ }
+ return c;
+}
+
+/* Retrieve the first numerical value contained in the string.
+ * Note: The value is supposed to be a 32-bit integer.
+ */
+static inline int get_next_value(const char __user **buf, size_t *len, int *val)
+{
+#define BUFLEN 11
+ int err, neg = 0, left;
+ char s[BUFLEN], *p;
+
+ err = skip_space(buf, len);
+ if (err < 0)
+  return err;
+ if (!*len)
+  return INT_MAX;
+ if (err == '\n') {
+  ++*buf;
+  --*len;
+  return INT_MAX;
+ }
+ if (err == '-') {
+  ++*buf;
+  --*len;
+  neg = 1;
+ }
+
+ left = *len;
+ if (left > sizeof(s) - 1)
+  left = sizeof(s) - 1;
+ if (copy_from_user(s, *buf, left))
+  return -EFAULT;
+
+ s[left] = 0;
+ p = s;
+ if (*p < '0' || *p > '9')
+  return -EINVAL;
+
+ *val = simple_strtoul(p, &p, 0);
+ if (neg)
+  *val = -(*val);
+
+ left = p-s;
+ (*len) -= left;
+ (*buf) += left;
+
+ return 0;
+#undef BUFLEN
+}
+
+/* semundo_readline: read a line of /proc/<PID>/semundo file
+ * Return the number of value read or an errcode
+ */
+static inline int semundo_readline(const char __user **buf, size_t *left,
+       int *id,  short *array, int array_len)
+{
+ int i, val, err;
+
+ /* Read semid */
+ err = get_next_value(buf, left, id);
+ if (err)
+  return err;
+
+ /* Read all (semundo-) values of a full line */
+ for (i = 0; ; i++) {
+
+  err = get_next_value(buf, left, &val);
+  if (err < 0)
+   return err;
+  /* reach end of line or end of buffer */
+  if (err == INT_MAX)
+   break;
+  /* Return an error if we get more values then expected */
+  if (i < array_len)
+   array[i] = val;
+  else
+   return -EINVAL;
+ }
+ return i;
+}
+
+/* semundo_update: set or update the undo values of the given undo_list
+ * for a given semaphore id.
+ */
+static inline int semundo_update(struct sem_undo_list *undo_list, int id,
+     short array[], int size)
+{
+ struct sem_undo *un;
+ struct sem_array *sma;
+ struct ipc_namespace *ns = undo_list->ns;
+
+retry_undo:
+ un = find_undo(undo_list, id);
+ if (IS_ERR(un))
+  return PTR_ERR(un);
+
+ /* lookup the sem_array */
+ sma = sem_lock(ns, id);
+ if (IS_ERR(sma))
+  return PTR_ERR(sma);
+
+ /*
+  * semid identifiers are not unique - find_undo may have
+  * allocated an undo structure, it was invalidated by an RMID
+  * and now a new array which received the same id.
+  * Check and retry.
+  */
+ if (un->semid == -1) {
+  sem_unlock(sma);
+  goto retry_undo;
+ }
```

```
+
+ /*
+  * If the number of values given does not match the number of
+  * semaphores in the array, consider this as an error.
+  */
+ if (size != sma->sem_nsems) {
+  sem_unlock(sma);
+  return -EINVAL;
+ }
+
+ /* update the undo values */
+ while (--size >= 0)
+  un->semadj[size] = array[size];
+
+ sem_unlock(sma);
+ return 0;
+}
+
+/*
+ * write operation for /proc/<pid>/semundo file
+ *
+ * It allows to set or update the sem_undo list for this task, for the given
+ * semaphore ID with the given 'undo' values.
+ *
+ * The expected format of the given string is
+ * "semID <val1> <val2> ... <valN>"
+ *
+ * The semID must match an existing semaphore array and the number of
+ * of values following the semID must match the number of semaphores in
+ * the corresponding array.
+ *
+ * Multiple semID can be passed simultaneously, the newline ('\n') behaving as
+ * a separator in this case.
+ *
+ * Note: passing a sem_undo list (all undo values of one semID) through multiple
+ * write-calls is not allowed.
+ */
+static ssize_t semundo_write(struct file *file, const char __user *buf,
+        size_t count, loff_t *ppos)
+{
+ struct seq_file *m = file->private_data;
+ short *array;
+ int err, max_sem, id = 0;
+ size_t left = count;
+ struct sem_undo_list *undo_list = m->private;
+
+ /* The undo_list must have been retrieved or created
+    in semundo_open() */
+ if (undo_list == NULL)
+  return -EINVAL;
+
+ max_sem = undo_list->ns->sc_semmsl;
+
+ array = kmalloc(sizeof(short)*max_sem, GFP_KERNEL);
+ if (array == NULL)
+  return -ENOMEM;
+
+ while (left) {
+  int nval;
+
+  /* Read a line */
+  nval = semundo_readline(&buf, &left, &id, array, max_sem);
+  if (nval < 0) {
+   err = nval;
+   goto out;
+  }
+
+  /* Update the values for the given semid */
+  err = semundo_update(undo_list, id, array, nval);
+  if (err)
+   goto out;
+ }
+ err = count - left;
+
+out:
+ kfree(array);
+ return err;
+}
+
 static int semundo_release(struct inode *inode, struct file *file)
 {
  struct seq_file *m = file->private_data;
@@ -1631,6 +1850,7 @@ static int semundo_release(struct inode
 const struct file_operations proc_semundo_operations = {
  .open  = semundo_open,
  .read  = seq_read,
+ .write  = semundo_write,
  .llseek  = seq_lseek,
  .release = semundo_release,
 };
```

--
Pierre Peiffer

https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID

Posted by Alexey Dobriyan on Tue, 29 Jan 2008 21:06:56 GMT

View Forum Message <> Reply to Message

On Tue, Jan 29, 2008 at 05:02:38PM +0100, pierre.peiffer@bull.net wrote:
> This patch provides three new API to change the ID of an existing
> System V IPCs.
>
> These APIs are:
> long msg_chid(struct ipc_namespace *ns, int id, int newid);
> long sem_chid(struct ipc_namespace *ns, int id, int newid);
> long shm_chid(struct ipc_namespace *ns, int id, int newid);
>
> They return 0 or an error code in case of failure.
>
> They may be useful for setting a specific ID for an IPC when preparing
> a restart operation.
>
> To be successful, the following rules must be respected:
> - the IPC exists (of course...)
> - the new ID must satisfy the ID computation rule.
> - the entry in the idr corresponding to the new ID must be free.

```
>  ipc/util.c        |  48 ++++++++++++++++++++++++++++++++++++++++++++++++
>  ipc/util.h        |   1 +
>  8 files changed, 197 insertions(+)
```

For the record, OpenVZ uses "create with predefined ID" method which
leads to less code. For example, change at the end is all we want from
ipc/util.c .

Also, if ids were A and B at the moment of checkpoint, and during
restart they became B and A you'll get collision in both ways which you
techically can avoid by classic "tmp = A, A = B, B = tmp" but you also
can avoid all other loops just by creating with ID you need.

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID

Posted by Pierre Peiffer on Wed, 30 Jan 2008 09:52:14 GMT
View Forum Message <> Reply to Message

Alexey Dobriyan wrote:
> On Tue, Jan 29, 2008 at 05:02:38PM +0100, pierre.peiffer@bull.net wrote:
>> This patch provides three new API to change the ID of an existing
>> System V IPCs.
>>
>> These APIs are:
>> long msg_chid(struct ipc_namespace *ns, int id, int newid);
>> long sem_chid(struct ipc_namespace *ns, int id, int newid);
>> long shm_chid(struct ipc_namespace *ns, int id, int newid);
>>
>> They return 0 or an error code in case of failure.
>>
>> They may be useful for setting a specific ID for an IPC when preparing
>> a restart operation.
>>
>> To be successful, the following rules must be respected:
>> - the IPC exists (of course...)
>> - the new ID must satisfy the ID computation rule.
>> - the entry in the idr corresponding to the new ID must be free.
>
>> ipc/util.c | 48 ++
>> ipc/util.h | 1 +
>> 8 files changed, 197 insertions(+)
>
> For the record, OpenVZ uses "create with predefined ID" method which
> leads to less code. For example, change at the end is all we want from
> ipc/util.c .
>

Yes, indeed, I saw that. The idea here is, at the end, to propose a more
"userspace oriented" solution.
As we can't use msgget(), etc, API to specify an ID, I think we can at least
change it afterwards

> Also, if ids were A and B at the moment of checkpoint, and during
> restart they became B and A you'll get collision in both ways which you
> techically can avoid by classic "tmp = A, A = B, B = tmp"

In the general case, yes, you're right.
In the case of the checkpoint/restart, this is not necessarily a problem, as we
will probably restart an application in an empty "container"/"namespace"; Thus
we can create all needed IPCs in an empty IPC namespace like this:
1. create first IPC
2. change its ID
3. create the second IPC
4. change its ID

5. etc..

But yes, I agree that if we can directly create an IPC with the right ID, it
would be better; may be with an IPC_CREATE command or something like that if the
direction is to do that from userspace.

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 12/15] (RFC) IPC/semaphores: make use of RCU to free the sem_undo_list
Posted by serue on Wed, 30 Jan 2008 21:26:50 GMT

View Forum Message <> Reply to Message

Quoting pierre.peiffer@bull.net (pierre.peiffer@bull.net):

```
> From: Pierre Peiffer <pierre.peiffer@bull.net>
>
> Today, the sem_undo_list is freed when the last task using it exits.
> There is no mechanism in place, that allows a safe concurrent access to
> the sem_undo_list of a target task and protects efficiently against a
> task-exit.
>
> That is okay for now as we don't need this.
>
> As I would like to provide a /proc interface to access this data, I need
> such a safe access, without blocking the target task if possible.
>
> This patch proposes to introduce the use of RCU to delay the real free of
> these sem_undo_list structures. They can then be accessed in a safe manner
> by any tasks inside read critical section, this way:
>
>  struct sem_undo_list *undo_list;
>  int ret;
>  ...
>  rcu_read_lock();
>  undo_list = rcu_dereference(task->sysvsem.undo_list);
>  if (undo_list)
>   ret = atomic_inc_not_zero(&undo_list->refcnt);
>  rcu_read_unlock();
>  ...
>  if (undo_list && ret) {
>   /* section where undo_list can be used quietly */
>   ...
>  }
>  ...
```

And of course then

```
 if (atomic_dec_and_test(&undo_list->refcnt))
  free_semundo_list(undo_list);
```

by that task.

```
>
> Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
```

Looks correct in terms of locking/refcounting.

Signed-off-by: Serge Hallyn <serue@us.ibm.com>

thanks,
-serge

```
> ---
>
>  include/linux/sem.h |    7 +++++--
>  ipc/sem.c           |   42 +++++++++++++++++++++++++----------------
>  2 files changed, 31 insertions(+), 18 deletions(-)
>
> Index: b/include/linux/sem.h
> ===================================================================
> --- a/include/linux/sem.h
> +++ b/include/linux/sem.h
> @@ -115,7 +115,8 @@ struct sem_queue {
>  };
>
>  /* Each task has a list of undo requests. They are executed automatically
> - * when the process exits.
> + * when the last refcnt of sem_undo_list is released (ie when the process exits
> + * in the general case)
>   */
>  struct sem_undo {
>   struct sem_undo * proc_next; /* next entry on this process */
> @@ -125,12 +126,14 @@ struct sem_undo {
>  };
>
>  /* sem_undo_list controls shared access to the list of sem_undo structures
> - * that may be shared among all a CLONE_SYSVSEM task group.
> + * that may be shared among all a CLONE_SYSVSEM task group or with an external
> + * process which changes the list through procfs.
>   */
>  struct sem_undo_list {
>   atomic_t refcnt;
>   spinlock_t lock;
>   struct sem_undo *proc_list;
> + struct ipc_namespace *ns;
>  };
>
>  struct sysv_sem {
> Index: b/ipc/sem.c
> ===================================================================
> --- a/ipc/sem.c
> +++ b/ipc/sem.c
> @@ -1038,6 +1038,7 @@ static inline int get_undo_list(struct s
>     return -ENOMEM;
>    spin_lock_init(&undo_list->lock);
>    atomic_set(&undo_list->refcnt, 1);
> +  undo_list->ns = get_ipc_ns(current->nsproxy->ipc_ns);
>    current->sysvsem.undo_list = undo_list;
>   }
>   *undo_listp = undo_list;
> @@ -1316,7 +1317,8 @@ int copy_semundo(unsigned long clone_fla
>  }
>
>  /*
> - * add semadj values to semaphores, free undo structures.
> + * add semadj values to semaphores, free undo structures, if there is no
> + * more user.
>   * undo structures are not freed when semaphore arrays are destroyed
>   * so some of them may be out of date.
>   * IMPLEMENTATION NOTE: There is some confusion over whether the
> @@ -1326,23 +1328,17 @@ int copy_semundo(unsigned long clone_fla
>   * The original implementation attempted to do this (queue and wait).
>   * The current implementation does not do so. The POSIX standard
>   * and SVID should be consulted to determine what behavior is mandated.
> + *
> + * Note:
> + * A concurrent task is only allowed to access and go through the list
> + * of sem_undo if it successfully grabs a refcnt.
>   */
> -void exit_sem(struct task_struct *tsk)
> +static void free_semundo_list(struct sem_undo_list *undo_list)
>  {
> - struct sem_undo_list *undo_list;
>   struct sem_undo *u, **up;
> - struct ipc_namespace *ns;
>
> - undo_list = tsk->sysvsem.undo_list;
> - if (!undo_list)
> -  return;
> -
> - 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.
> + /* There's no need to hold the semundo list lock, as there are
> +  * no more tasks or possible users for this undo list.
>    */
>   for (up = &undo_list->proc_list; (u = *up); *up = u->proc_next, kfree(u)) {
>    struct sem_array *sma;
> @@ -1354,7 +1350,7 @@ void exit_sem(struct task_struct *tsk)
>
>    if(semid == -1)
>     continue;
> -  sma = sem_lock(ns, semid);
> +  sma = sem_lock(undo_list->ns, semid);
>    if (IS_ERR(sma))
>     continue;
>
> @@ -1368,7 +1364,8 @@ void exit_sem(struct task_struct *tsk)
>     if (u == un)
>      goto found;
>    }
> -  printk ("exit_sem undo list error id=%d\n", u->semid);
> +  printk(KERN_ERR "free_semundo_list error id=%d\n",
> +         u->semid);
>    goto next_entry;
>  found:
>    *unp = un->id_next;
> @@ -1404,9 +1401,22 @@ found:
>  next_entry:
>    sem_unlock(sma);
>   }
> + put_ipc_ns(undo_list->ns);
>   kfree(undo_list);
>  }
>
> +/* called from do_exit() */
> +void exit_sem(struct task_struct *tsk)
> +{
> + struct sem_undo_list *ul = tsk->sysvsem.undo_list;
> + if (ul) {
> +  rcu_assign_pointer(tsk->sysvsem.undo_list, NULL);
> +  synchronize_rcu();
> +  if (atomic_dec_and_test(&ul->refcnt))
> +	free_semundo_list(ul);
> +	}
> +}
> +
>  #ifdef CONFIG_PROC_FS
>  static int sysvipc_sem_proc_show(struct seq_file *s, void *it)
>  {
>
> --
> Pierre Peiffer
> _______________________________________________
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers
```

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 14/15] (RFC) IPC/semaphores: prepare semundo code to work on another task
Posted by serue on Wed, 30 Jan 2008 21:44:30 GMT

View Forum Message <> Reply to Message

Quoting pierre.peiffer@bull.net (pierre.peiffer@bull.net):

```
> From: Pierre Peiffer <pierre.peiffer@bull.net>
>
> In order to modify the semundo-list of a task from procfs, we must be able to
> work on any target task.
> But all the existing code playing with the semundo-list, currently works
> only on the 'current' task, and does not allow to specify any target task.
>
> This patch changes all these routines to allow them to work on a specified
> task, passed in parameter, instead of current.
>
> This is mainly a preparation for the semundo_write() operation, on the
> /proc/<pid>/semundo file, as provided in the next patch.
>
> Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
> ---
>
>  ipc/sem.c |   90 ++++++++++++++++++++++++++++++++++++++++++++++---------------
>  1 file changed, 68 insertions(+), 22 deletions(-)
>
> Index: b/ipc/sem.c
> ===================================================================
> --- a/ipc/sem.c
> +++ b/ipc/sem.c
> @@ -1017,8 +1017,9 @@ asmlinkage long sys_semctl (int semid, i
>  }
>
>  /* If the task doesn't already have a undo_list, then allocate one
> - * here.  We guarantee there is only one thread using this undo list,
> - * and current is THE ONE
> + * here.
> + * The target task (tsk) is current in the general case, except when
> + * accessed from the procfs (ie when writting to /proc/<pid>/semundo)
>   *
>   * If this allocation and assignment succeeds, but later
>   * portions of this code fail, there is no need to free the sem_undo_list.
> @@ -1026,22 +1027,60 @@ asmlinkage long sys_semctl (int semid, i
>   * at exit time.
>   *
>   * This can block, so callers must hold no locks.
> + *
> + * Note: task_lock is used to synchronize 1. several possible concurrent
> + * creations and 2. the free of the undo_list (done when the task using it
> + * exits). In the second case, we check the PF_EXITING flag to not create
> + * an undo_list for a task which has exited.
> + * If there already is an undo_list for this task, there is no need
> + * to held the task-lock to retrieve it, as the pointer can not change
> + * afterwards.
>   */
> -static inline int get_undo_list(struct sem_undo_list **undo_listp)
> +static inline int get_undo_list(struct task_struct *tsk,
> +    struct sem_undo_list **ulp)
>  {
> - struct sem_undo_list *undo_list;
> + if (tsk->sysvsem.undo_list == NULL) {
> +  struct sem_undo_list *undo_list;
```

Hmm, this is weird. If there was no undo_list and
tsk!=current, you set the refcnt to 2. But if there was an
undo list and tsk!=current, where do you inc the refcnt?

```
>
> - undo_list = current->sysvsem.undo_list;
> - if (!undo_list) {
> -  undo_list = kzalloc(sizeof(*undo_list), GFP_KERNEL);
> +  /* we must alloc a new one */
> +  undo_list = kmalloc(sizeof(*undo_list), GFP_KERNEL);
>    if (undo_list == NULL)
>     return -ENOMEM;
> +
> +  task_lock(tsk);
> +
> +  /* check again if there is an undo_list for this task */
> +  if (tsk->sysvsem.undo_list) {
> +   if (tsk != current)
> +    atomic_inc(&tsk->sysvsem.undo_list->refcnt);
> +   task_unlock(tsk);
> +   kfree(undo_list);
> +   goto out;
> +  }
> +
>    spin_lock_init(&undo_list->lock);
> -  atomic_set(&undo_list->refcnt, 1);
> -  undo_list->ns = get_ipc_ns(current->nsproxy->ipc_ns);
> -  current->sysvsem.undo_list = undo_list;
> +  /*
> +   * If tsk is not current (meaning that current is creating
> +   * a semundo_list for a target task through procfs), and if
> +   * it's not being exited then refcnt must be 2: the target
> +   * task tsk + current.
> +   */
> +  if (tsk == current)
> +   atomic_set(&undo_list->refcnt, 1);
> +  else if (!(tsk->flags & PF_EXITING))
> +   atomic_set(&undo_list->refcnt, 2);
> +  else {
> +   task_unlock(tsk);
> +   kfree(undo_list);
> +   return -EINVAL;
> +  }
> +  undo_list->ns = get_ipc_ns(tsk->nsproxy->ipc_ns);
> +  undo_list->proc_list = NULL;
> +  tsk->sysvsem.undo_list = undo_list;
> +  task_unlock(tsk);
>   }
> - *undo_listp = undo_list;
> +out:
> + *ulp = tsk->sysvsem.undo_list;
>   return 0;
>  }
>
> @@ -1065,17 +1104,12 @@ static struct sem_undo *lookup_undo(stru
>   return un;
>  }
>
> -static struct sem_undo *find_undo(struct ipc_namespace *ns, int semid)
> +static struct sem_undo *find_undo(struct sem_undo_list *ulp, int semid)
>  {
>   struct sem_array *sma;
> - struct sem_undo_list *ulp;
>   struct sem_undo *un, *new;
> + struct ipc_namespace *ns;
>   int nsems;
> - int error;
> -
> - error = get_undo_list(&ulp);
> - if (error)
> -  return ERR_PTR(error);
>
>   spin_lock(&ulp->lock);
>   un = lookup_undo(ulp, semid);
> @@ -1083,6 +1117,8 @@ static struct sem_undo *find_undo(struct
>   if (likely(un!=NULL))
>    goto out;
>
> + ns = ulp->ns;
> +
>   /* no undo structure around - allocate one. */
>   sma = sem_lock_check(ns, semid);
>   if (IS_ERR(sma))
> @@ -1133,6 +1169,7 @@ asmlinkage long sys_semtimedop(int semid
>   struct sem_array *sma;
>   struct sembuf fast_sops[SEMOPM_FAST];
>   struct sembuf* sops = fast_sops, *sop;
> + struct sem_undo_list *ulp;
>   struct sem_undo *un;
>   int undos = 0, alter = 0, max;
>   struct sem_queue queue;
> @@ -1177,9 +1214,13 @@ asmlinkage long sys_semtimedop(int semid
>     alter = 1;
>   }
>
> + error = get_undo_list(current, &ulp);
> + if (error)
> +  goto out_free;
> +
>  retry_undos:
>   if (undos) {
> -  un = find_undo(ns, semid);
> +  un = find_undo(ulp, semid);
>    if (IS_ERR(un)) {
>     error = PTR_ERR(un);
>     goto out_free;
> @@ -1305,7 +1346,7 @@ int copy_semundo(unsigned long clone_fla
>   int error;
>
>   if (clone_flags & CLONE_SYSVSEM) {
```

```
> -  error = get_undo_list(&undo_list);
> +  error = get_undo_list(current, &undo_list);
>    if (error)
>     return error;
>    atomic_inc(&undo_list->refcnt);
> @@ -1405,10 +1446,15 @@ next_entry:
>   kfree(undo_list);
>  }
>
> -/* called from do_exit() */
> +/* exit_sem: called from do_exit()
> + * task_lock is used to synchronize with get_undo_list()
```

Ok I had to think about this again. I'd like the comment here to point out that the task_lock here acts as a barrier between the prior setting of PF_EXITING and the undo_list being freed here, so that get_undo_list() will either see PF_EXITING is NOT in the tsk->flags, in which case it will insert the undo_list before the task_lock() is grabbed here, and with count=2, so that it gets correctly put here in exit_sem, or it will see PF_EXITING set and cancel the undo_list it was creating.

```
> + */
>  void exit_sem(struct task_struct *tsk)
>  {
> - struct sem_undo_list *ul = tsk->sysvsem.undo_list;
> + struct sem_undo_list *ul;
> + task_lock(tsk);
> + ul = tsk->sysvsem.undo_list;
> + task_unlock(tsk);
>   if (ul) {
>    rcu_assign_pointer(tsk->sysvsem.undo_list, NULL);
>    synchronize_rcu();
>
```

> --
> Pierre Peiffer
> ____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 05/15] IPC/semaphores: remove one unused parameter from semctl_down()

Posted by Nadia Derbey on Thu, 31 Jan 2008 08:32:46 GMT

View Forum Message <> Reply to Message

pierre.peiffer@bull.net wrote:

```
> From: Pierre Peiffer <pierre.peiffer@bull.net>
> 
> semctl_down() takes one unused parameter: semnum.
> This patch proposes to get rid of it.
> 
> Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
> Acked-by: Serge Hallyn <serue@us.ibm.com>
> ---
>  ipc/sem.c |    6 +++---
>  1 file changed, 3 insertions(+), 3 deletions(-)
> 
> Index: b/ipc/sem.c
> ===================================================================
> --- a/ipc/sem.c
> +++ b/ipc/sem.c
> @@ -882,8 +882,8 @@ static inline unsigned long copy_semid_f
>   * to be held in write mode.
>   * NOTE: no locks must be held, the rw_mutex is taken inside this function.
>   */
> -static int semctl_down(struct ipc_namespace *ns, int semid, int semnum,
> -  int cmd, int version, union semun arg)
> +static int semctl_down(struct ipc_namespace *ns, int semid,
> +         int cmd, int version, union semun arg)
>  {
>   struct sem_array *sma;
>   int err;
> @@ -974,7 +974,7 @@ asmlinkage long sys_semctl (int semid, i
>    return err;
>   case IPC_RMID:
>   case IPC_SET:
> -  err = semctl_down(ns,semid,semnum,cmd,version,arg);
> +  err = semctl_down(ns, semid, cmd, version, arg);
>    return err;
>   default:
>    return -EINVAL;
> 
```

Looks like semnum is only used in semctl_main(). Why not removing it
from semctl_nolock() too?

Regards,
Nadia

Subject: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID
Posted by Pierre Peiffer on Thu, 31 Jan 2008 09:00:55 GMT

View Forum Message <> Reply to Message

Hi again,

Thinking more about this, I think I must clarify why I choose this way.
In fact, the idea of these patches is to provide the missing user APIs (or
extend the existing ones) that allow to set or update _all_ properties of all
IPCs, as needed in the case of the checkpoint/restart of an application (the
current user API does not allow to specify an ID for a created IPC, for
example). And this, without changing the existing API of course.

And msgget(), semget() and shmget() does not have any parameter we can use to
specify an ID.
That's why I've decided to not change these routines and add a new control
command, IP_SETID, with which we can can change the ID of an IPC. (that looks to
me more straightforward and logical)

Now, this patch is, in fact, only a preparation for the patch 10/15 which
really complete the user API by adding this IPC_SETID command.

(... continuing below ...)

Alexey Dobriyan wrote:
> On Tue, Jan 29, 2008 at 05:02:38PM +0100, pierre.peiffer@bull.net wrote:
>> This patch provides three new API to change the ID of an existing
>> System V IPCs.
>>
>> These APIs are:
>> long msg_chid(struct ipc_namespace *ns, int id, int newid);
>> long sem_chid(struct ipc_namespace *ns, int id, int newid);
>> long shm_chid(struct ipc_namespace *ns, int id, int newid);
>>
>> They return 0 or an error code in case of failure.
>>
>> They may be useful for setting a specific ID for an IPC when preparing
>> a restart operation.
>>

>> To be successful, the following rules must be respected:
>> - the IPC exists (of course...)
>> - the new ID must satisfy the ID computation rule.
>> - the entry in the idr corresponding to the new ID must be free.
>
>> ipc/util.c | 48 ++
>> ipc/util.h | 1 +
>> 8 files changed, 197 insertions(+)
>
> For the record, OpenVZ uses "create with predefined ID" method which
> leads to less code. For example, change at the end is all we want from
> ipc/util.c .

And in fact, you do that from kernel space, you don't have the constraint to fit
the existing user API.
Again, this patch, even if it presents a new kernel API, is in fact a
preparation for the next patch which introduces a new user API.

Do you think that this could fit your need ?

--
Pierre

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 2.6.24-rc8-mm1 14/15] (RFC) IPC/semaphores: prepare semundo code to work on another task
Posted by Pierre Peiffer on Thu, 31 Jan 2008 09:48:56 GMT

View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting pierre.peiffer@bull.net (pierre.peiffer@bull.net):
>> From: Pierre Peiffer <pierre.peiffer@bull.net>
>>
>> In order to modify the semundo-list of a task from procfs, we must be able to
>> work on any target task.
>> But all the existing code playing with the semundo-list, currently works
>> only on the 'current' task, and does not allow to specify any target task.
>>
>> This patch changes all these routines to allow them to work on a specified
>> task, passed in parameter, instead of current.
>>
>> This is mainly a preparation for the semundo_write() operation, on the
>> /proc/<pid>/semundo file, as provided in the next patch.
>>

```
>> Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
>> ---
>>
>>  ipc/sem.c |   90 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++----------------------
>>  1 file changed, 68 insertions(+), 22 deletions(-)
>>
>> Index: b/ipc/sem.c
>> ===================================================================
>> --- a/ipc/sem.c
>> +++ b/ipc/sem.c
>> @@ -1017,8 +1017,9 @@ asmlinkage long sys_semctl (int semid, i
>>  }
>>
>>  /* If the task doesn't already have a undo_list, then allocate one
>> - * here.  We guarantee there is only one thread using this undo list,
>> - * and current is THE ONE
>> + * here.
>> + * The target task (tsk) is current in the general case, except when
>> + * accessed from the procfs (ie when writting to /proc/<pid>/semundo)
>>   *
>>   * If this allocation and assignment succeeds, but later
>>   * portions of this code fail, there is no need to free the sem_undo_list.
>> @@ -1026,22 +1027,60 @@ asmlinkage long sys_semctl (int semid, i
>>   * at exit time.
>>   *
>>   * This can block, so callers must hold no locks.
>> + *
>> + * Note: task_lock is used to synchronize 1. several possible concurrent
>> + * creations and 2. the free of the undo_list (done when the task using it
>> + * exits). In the second case, we check the PF_EXITING flag to not create
>> + * an undo_list for a task which has exited.
>> + * If there already is an undo_list for this task, there is no need
>> + * to held the task-lock to retrieve it, as the pointer can not change
>> + * afterwards.
>>   */
>> -static inline int get_undo_list(struct sem_undo_list **undo_listp)
>> +static inline int get_undo_list(struct task_struct *tsk,
>> +    struct sem_undo_list **ulp)
>>  {
>> - struct sem_undo_list *undo_list;
>> + if (tsk->sysvsem.undo_list == NULL) {
>> +  struct sem_undo_list *undo_list;
>
> Hmm, this is weird.  If there was no undo_list and
> tsk!=current, you set the refcnt to 2.  But if there was an
> undo list and tsk!=current, where do you inc the refcnt?
>
```

I inc it outside this function, as I don't call get_undo_list() if there is an undo_list.
This appears most clearly in the next patch, in semundo_open() for example.

```
>> - undo_list = current->sysvsem.undo_list;
>> - if (!undo_list) {
>> -  undo_list = kzalloc(sizeof(*undo_list), GFP_KERNEL);
>> +  /* we must alloc a new one */
>> +  undo_list = kmalloc(sizeof(*undo_list), GFP_KERNEL);
>>    if (undo_list == NULL)
>>     return -ENOMEM;
>> +
>> +  task_lock(tsk);
>> +
>> +  /* check again if there is an undo_list for this task */
>> +  if (tsk->sysvsem.undo_list) {
>> +   if (tsk != current)
>> +    atomic_inc(&tsk->sysvsem.undo_list->refcnt);
>> +   task_unlock(tsk);
>> +   kfree(undo_list);
>> +   goto out;
>> +  }
>> +
>>    spin_lock_init(&undo_list->lock);
>> -  atomic_set(&undo_list->refcnt, 1);
>> -  undo_list->ns = get_ipc_ns(current->nsproxy->ipc_ns);
>> -  current->sysvsem.undo_list = undo_list;
>> +  /*
>> +   * If tsk is not current (meaning that current is creating
>> +   * a semundo_list for a target task through procfs), and if
>> +   * it's not being exited then refcnt must be 2: the target
>> +   * task tsk + current.
>> +   */
>> +  if (tsk == current)
>> +   atomic_set(&undo_list->refcnt, 1);
>> +  else if (!(tsk->flags & PF_EXITING))
>> +   atomic_set(&undo_list->refcnt, 2);
>> +  else {
>> +   task_unlock(tsk);
>> +   kfree(undo_list);
>> +   return -EINVAL;
>> +  }
>> +  undo_list->ns = get_ipc_ns(tsk->nsproxy->ipc_ns);
>> +  undo_list->proc_list = NULL;
>> +  tsk->sysvsem.undo_list = undo_list;
>> +  task_unlock(tsk);
>>   }
>> - *undo_listp = undo_list;
>> +out:
>> + *ulp = tsk->sysvsem.undo_list;
>>   return 0;
>>  }
>>
>> @@ -1065,17 +1104,12 @@ static struct sem_undo *lookup_undo(stru
>>   return un;
>>  }
>>
>> -static struct sem_undo *find_undo(struct ipc_namespace *ns, int semid)
>> +static struct sem_undo *find_undo(struct sem_undo_list *ulp, int semid)
>>  {
>>   struct sem_array *sma;
>> - struct sem_undo_list *ulp;
>>   struct sem_undo *un, *new;
>> + struct ipc_namespace *ns;
>>   int nsems;
>> - int error;
>> -
>> - error = get_undo_list(&ulp);
>> - if (error)
>> -  return ERR_PTR(error);
>>
>>   spin_lock(&ulp->lock);
>>   un = lookup_undo(ulp, semid);
>> @@ -1083,6 +1117,8 @@ static struct sem_undo *find_undo(struct
>>   if (likely(un!=NULL))
>>    goto out;
>>
>> + ns = ulp->ns;
>> +
>>   /* no undo structure around - allocate one. */
>>   sma = sem_lock_check(ns, semid);
>>   if (IS_ERR(sma))
>> @@ -1133,6 +1169,7 @@ asmlinkage long sys_semtimedop(int semid
>>   struct sem_array *sma;
>>   struct sembuf fast_sops[SEMOPM_FAST];
>>   struct sembuf* sops = fast_sops, *sop;
>> + struct sem_undo_list *ulp;
>>   struct sem_undo *un;
>>   int undos = 0, alter = 0, max;
>>   struct sem_queue queue;
>> @@ -1177,9 +1214,13 @@ asmlinkage long sys_semtimedop(int semid
>>     alter = 1;
>>   }
>>
>> + error = get_undo_list(current, &ulp);
>> + if (error)
>> +  goto out_free;
>> +
>>  retry_undos:
>>   if (undos) {
>> -  un = find_undo(ns, semid);
>> +  un = find_undo(ulp, semid);
>>    if (IS_ERR(un)) {
>>     error = PTR_ERR(un);
>>     goto out_free;
>> @@ -1305,7 +1346,7 @@ int copy_semundo(unsigned long clone_fla
>>   int error;
>>
>>   if (clone_flags & CLONE_SYSVSEM) {
>> -  error = get_undo_list(&undo_list);
>> +  error = get_undo_list(current, &undo_list);
>>    if (error)
>>     return error;
>>    atomic_inc(&undo_list->refcnt);
>> @@ -1405,10 +1446,15 @@ next_entry:
>>   kfree(undo_list);
>>  }
>>
>> -/* called from do_exit() */
>> +/* exit_sem: called from do_exit()
>> + * task_lock is used to synchronize with get_undo_list()
>
> Ok I had to think about this again.  I'd like the comment
> here to point out that the task_lock here acts as a barrier
> between the prior setting of PF_EXITING and the undo_list
> being freed here, so that get_undo_list() will either see
> PF_EXITING is NOT in the tsk->flags, in which case it will
> insert the undo_list before the task_lock() is grabbed here,
> and with count=2, so that it gets correctly put here in
> exit_sem, or it will see PF_EXITING set and cancel the
> undo_list it was creating.
>
```

Yep, I will add this to clarify this point.

Thanks Serge.

P.

```
>> + */
>>  void exit_sem(struct task_struct *tsk)
>>  {
>> - struct sem_undo_list *ul = tsk->sysvsem.undo_list;
>> + struct sem_undo_list *ul;
```

>> + task_lock(tsk);
>> + ul = tsk->sysvsem.undo_list;
>> + task_unlock(tsk);
>>   if (ul) {
>>   rcu_assign_pointer(tsk->sysvsem.undo_list, NULL);
>>   synchronize_rcu();
>>
>> --
>> Pierre Peiffer
>> ____
>> Containers mailing list
>> Containers@lists.linux-foundation.org
>> https://lists.linux-foundation.org/mailman/listinfo/containers
>
>

--
Pierre Peiffer
____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 12/15] (RFC) IPC/semaphores: make use of RCU to free the sem_undo_list
Posted by Pierre Peiffer on Thu, 31 Jan 2008 09:52:08 GMT

View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting pierre.peiffer@bull.net (pierre.peiffer@bull.net):
>> From: Pierre Peiffer <pierre.peiffer@bull.net>
>>
>> Today, the sem_undo_list is freed when the last task using it exits.
>> There is no mechanism in place, that allows a safe concurrent access to
>> the sem_undo_list of a target task and protects efficiently against a
>> task-exit.
>>
>> That is okay for now as we don't need this.
>>
>> As I would like to provide a /proc interface to access this data, I need
>> such a safe access, without blocking the target task if possible.
>>
>> This patch proposes to introduce the use of RCU to delay the real free of
>> these sem_undo_list structures. They can then be accessed in a safe manner
>> by any tasks inside read critical section, this way:
>>
>> struct sem_undo_list *undo_list;

```
>>  int ret;
>>  ...
>>  rcu_read_lock();
>>  undo_list = rcu_dereference(task->sysvsem.undo_list);
>>  if (undo_list)
>>   ret = atomic_inc_not_zero(&undo_list->refcnt);
>>  rcu_read_unlock();
>>  ...
>>  if (undo_list && ret) {
>>   /* section where undo_list can be used quietly */
>>   ...
>>  }
>>  ...
>
> And of course then
>
>  if (atomic_dec_and_test(&undo_list->refcnt))
>   free_semundo_list(undo_list);
>
> by that task.
>
```

I will precise this too.

```
>> Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
>
> Looks correct in terms of locking/refcounting.
>
> Signed-off-by: Serge Hallyn <serue@us.ibm.com>
>
```

Thanks !

--
Pierre

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID

Posted by dev on Thu, 31 Jan 2008 09:54:10 GMT

View Forum Message <> Reply to Message

Why user space can need this API? for checkpointing only?

Then I would not consider it for inclusion until it is clear how to implement checkpointing.

As for me personally - I'm against exporting such APIs, since they are not needed in real-life user space applications and maintaining it forever for compatibility doesn't worth it.
Also such APIs allow creation of non-GPL checkpointing in user-space, which can be of concern as well.

Kirill

Pierre Peiffer wrote:
> Hi again,
>
> Thinking more about this, I think I must clarify why I choose this way.
> In fact, the idea of these patches is to provide the missing user APIs (or
> extend the existing ones) that allow to set or update _all_ properties of all
> IPCs, as needed in the case of the checkpoint/restart of an application (the
> current user API does not allow to specify an ID for a created IPC, for
> example). And this, without changing the existing API of course.
>
> And msgget(), semget() and shmget() does not have any parameter we can use to
> specify an ID.
> That's why I've decided to not change these routines and add a new control
> command, IP_SETID, with which we can can change the ID of an IPC. (that looks to
> me more straightforward and logical)
>
> Now, this patch is, in fact, only a preparation for the patch 10/15 which
> really complete the user API by adding this IPC_SETID command.
>
> (... continuing below ...)
>
> Alexey Dobriyan wrote:
>> On Tue, Jan 29, 2008 at 05:02:38PM +0100, pierre.peiffer@bull.net wrote:
>>> This patch provides three new API to change the ID of an existing
>>> System V IPCs.
>>>
>>> These APIs are:
>>> long msg_chid(struct ipc_namespace *ns, int id, int newid);
>>> long sem_chid(struct ipc_namespace *ns, int id, int newid);
>>> long shm_chid(struct ipc_namespace *ns, int id, int newid);
>>>
>>> They return 0 or an error code in case of failure.
>>>
>>> They may be useful for setting a specific ID for an IPC when preparing
>>> a restart operation.
>>>
>>> To be successful, the following rules must be respected:
>>> - the IPC exists (of course...)

>>> - the new ID must satisfy the ID computation rule.
>>> - the entry in the idr corresponding to the new ID must be free.
>>> ipc/util.c | 48 ++
>>> ipc/util.h | 1 +
>>> 8 files changed, 197 insertions(+)
>> For the record, OpenVZ uses "create with predefined ID" method which
>> leads to less code. For example, change at the end is all we want from
>> ipc/util.c .
>
> And in fact, you do that from kernel space, you don't have the constraint to fit
> the existing user API.
> Again, this patch, even if it presents a new kernel API, is in fact a
> preparation for the next patch which introduces a new user API.
>
> Do you think that this could fit your need ?
>

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 05/15] IPC/semaphores: remove one unused parameter from semctl_down()

Posted by Pierre Peiffer on Thu, 31 Jan 2008 10:18:30 GMT

View Forum Message <> Reply to Message

Nadia Derbey wrote:
> pierre.peiffer@bull.net wrote:
>> From: Pierre Peiffer <pierre.peiffer@bull.net>
>>
>> semctl_down() takes one unused parameter: semnum.
>> This patch proposes to get rid of it.
>>
>> Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
>> Acked-by: Serge Hallyn <serue@us.ibm.com>
>> ---
>> ipc/sem.c | 6 +++---
>> 1 file changed, 3 insertions(+), 3 deletions(-)
>>
>> Index: b/ipc/sem.c
>> ===
>> --- a/ipc/sem.c
>> +++ b/ipc/sem.c
>> @@ -882,8 +882,8 @@ static inline unsigned long copy_semid_f
>>   * to be held in write mode.
>>   * NOTE: no locks must be held, the rw_mutex is taken inside this
>> function.

```
>>   */
>> -static int semctl_down(struct ipc_namespace *ns, int semid, int semnum,
>> -        int cmd, int version, union semun arg)
>> +static int semctl_down(struct ipc_namespace *ns, int semid,
>> +               int cmd, int version, union semun arg)
>>  {
>>      struct sem_array *sma;
>>      int err;
>> @@ -974,7 +974,7 @@ asmlinkage long sys_semctl (int semid, i
>>          return err;
>>      case IPC_RMID:
>>      case IPC_SET:
>> -        err = semctl_down(ns,semid,semnum,cmd,version,arg);
>> +        err = semctl_down(ns, semid, cmd, version, arg);
>>          return err;
>>      default:
>>          return -EINVAL;
>>
>
```

> Looks like semnum is only used in semctl_main(). Why not removing it
> from semctl_nolock() too?

Indeed.
In fact, I already fixed that in a previous patch, included in -mm since kernel
2.6.24.rc3-mm2 (patch named ipc-semaphores-consolidate-sem_stat-and.patch)

--
Pierre

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 05/15] IPC/semaphores: remove one unused parameter from semctl_down()
Posted by Nadia Derbey on Thu, 31 Jan 2008 11:30:51 GMT
View Forum Message <> Reply to Message

Pierre Peiffer wrote:
>
> Nadia Derbey wrote:
>
>>pierre.peiffer@bull.net wrote:
>>
>>>From: Pierre Peiffer <pierre.peiffer@bull.net>
>>>
>>>semctl_down() takes one unused parameter: semnum.

```
>>>This patch proposes to get rid of it.
>>>
>>>Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
>>>Acked-by: Serge Hallyn <serue@us.ibm.com>
>>>---
>>> ipc/sem.c |    6 +++---
>>> 1 file changed, 3 insertions(+), 3 deletions(-)
>>>
>>>Index: b/ipc/sem.c
>>>===================================================================
>>>--- a/ipc/sem.c
>>>+++ b/ipc/sem.c
>>>@@ -882,8 +882,8 @@ static inline unsigned long copy_semid_f
>>>  * to be held in write mode.
>>>  * NOTE: no locks must be held, the rw_mutex is taken inside this
>>>function.
>>>  */
>>>-static int semctl_down(struct ipc_namespace *ns, int semid, int semnum,
>>>-	int cmd, int version, union semun arg)
>>>+static int semctl_down(struct ipc_namespace *ns, int semid,
>>>+		       int cmd, int version, union semun arg)
>>> {
>>> 	struct sem_array *sma;
>>> 	int err;
>>>@@ -974,7 +974,7 @@ asmlinkage long sys_semctl (int semid, i
>>> 		return err;
>>> 	case IPC_RMID:
>>> 	case IPC_SET:
>>>-		err = semctl_down(ns,semid,semnum,cmd,version,arg);
>>>+		err = semctl_down(ns, semid, cmd, version, arg);
>>> 		return err;
>>> 	default:
>>> 		return -EINVAL;
>>>
>>
>>Looks like semnum is only used in semctl_main(). Why not removing it
>>from semctl_nolock() too?
>
>
> Indeed.
> In fact, I already fixed that in a previous patch, included in -mm since kernel
> 2.6.24.rc3-mm2 (patch named ipc-semaphores-consolidate-sem_stat-and.patch)
>
```

Oops.. Sorry for the "noise"!

____
Containers mailing list
Containers@lists.linux-foundation.org

https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID
Posted by Pierre Peiffer on Thu, 31 Jan 2008 11:57:00 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
> Why user space can need this API? for checkpointing only?

I would say "at least for checkpointing"... ;) May be someone else may find an
interest about this for something else.
In fact, I'm sure that you have some interest in checkpointing; and thus, you
have probably some ideas in mind; but whatever the solution you will propose,
I'm pretty sure that I could say the same thing for your solution.
And what I finally think is: even if it's for "checkpointing only", if many
people are interested by this, it may be sufficient to push this ?

> Then I would not consider it for inclusion until it is clear how to implement checkpointing.
> As for me personally - I'm against exporting such APIs, since they are not needed in real-life user space applications and maintaining it forever for compatibility doesn't worth it.

Maintaining these patches is not a big deal, really, but this is not the main
point; the "need in real life" (1) is in fact the main one, and then, the "is
this solution the best one ?" (2) the second one.

About (1), as said in my first mail, as the namespaces and containers are being
integrated into the mainline kernel, checkpoint/restart is (or will be) the next
need.
About (2), my solution propose to do that, as much as possible from userspace,
to minimize the kernel impact. Of course, this is subject to discussion. My
opinion is that doing a full checkpoint/restart from kernel space will need lot
of new specific and intrusive code; I'm not sure that this will be acceptable by
the community. But this is my opinion only. Discusion is opened.

> Also such APIs allow creation of non-GPL checkpointing in user-space, which can be of concern as well.

Honestly, I don't think this really a concern at all. I mean: I've never seen
"this allows non-GPL binary and thus, this is bad" as an argument to reject a
functionality, but I may be wrong, and thus, it can be discussed as well.
I think the points (1) and (2) as stated above are the key ones.

Pierre

> Kirill
>

>
> Pierre Peiffer wrote:
>> Hi again,
>>
>> Thinking more about this, I think I must clarify why I choose this way.
>> In fact, the idea of these patches is to provide the missing user APIs (or
>> extend the existing ones) that allow to set or update _all_ properties of all
>> IPCs, as needed in the case of the checkpoint/restart of an application (the
>> current user API does not allow to specify an ID for a created IPC, for
>> example). And this, without changing the existing API of course.
>>
>> And msgget(), semget() and shmget() does not have any parameter we can use to
>> specify an ID.
>> That's why I've decided to not change these routines and add a new control
>> command, IP_SETID, with which we can can change the ID of an IPC. (that looks to
>> me more straightforward and logical)
>>
>> Now, this patch is, in fact, only a preparation for the patch 10/15 which
>> really complete the user API by adding this IPC_SETID command.
>>
>> (... continuing below ...)
>>
>> Alexey Dobriyan wrote:
>>> On Tue, Jan 29, 2008 at 05:02:38PM +0100, pierre.peiffer@bull.net wrote:
>>>> This patch provides three new API to change the ID of an existing
>>>> System V IPCs.
>>>>
>>>> These APIs are:
>>>> long msg_chid(struct ipc_namespace *ns, int id, int newid);
>>>> long sem_chid(struct ipc_namespace *ns, int id, int newid);
>>>> long shm_chid(struct ipc_namespace *ns, int id, int newid);
>>>>
>>>> They return 0 or an error code in case of failure.
>>>>
>>>> They may be useful for setting a specific ID for an IPC when preparing
>>>> a restart operation.
>>>>
>>>> To be successful, the following rules must be respected:
>>>> - the IPC exists (of course...)
>>>> - the new ID must satisfy the ID computation rule.
>>>> - the entry in the idr corresponding to the new ID must be free.
>>>> ipc/util.c | 48 ++
>>>> ipc/util.h | 1 +
>>>> 8 files changed, 197 insertions(+)
>>> For the record, OpenVZ uses "create with predefined ID" method which
>>> leads to less code. For example, change at the end is all we want from
>>> ipc/util.c .
>> And in fact, you do that from kernel space, you don't have the constraint to fit

>> the existing user API.
>> Again, this patch, even if it presents a new kernel API, is in fact a
>> preparation for the next patch which introduces a new user API.
>>
>> Do you think that this could fit your need ?
>>
>
>

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID
Posted by dev on Thu, 31 Jan 2008 13:11:26 GMT

View Forum Message <> Reply to Message

Pierre,

my point is that after you've added interface "set IPCID", you'll need more and more for checkpointing:
- "create/setup conntrack" (otherwise connections get dropped),
- "set task start time" (needed for Oracle checkpointing BTW),
- "set some statistics counters (e.g. networking or taskstats)"
- "restore inotify"
and so on and so forth.

Exporting such intimate kernel interfaces to user space doesn't look sane.
Exactly from compatibility and maintenance POV. You'll be burden with supporting them for a long time.
Remember recent story with SLUB and /proc/slabinfo?

Hope I made my argument more clear this time.

Thanks,
Kirill

Pierre Peiffer wrote:
>
> Kirill Korotaev wrote:
>> Why user space can need this API? for checkpointing only?
>

> I would say "at least for checkpointing"... ;) May be someone else may find an
> interest about this for something else.
> In fact, I'm sure that you have some interest in checkpointing; and thus, you
> have probably some ideas in mind; but whatever the solution you will propose,
> I'm pretty sure that I could say the same thing for your solution.
> And what I finally think is: even if it's for "checkpointing only", if many
> people are interested by this, it may be sufficient to push this ?
>
>> Then I would not consider it for inclusion until it is clear how to implement checkpointing.
>> As for me personally - I'm against exporting such APIs, since they are not needed in real-life user space applications and maintaining it forever for compatibility doesn't worth it.
>
> Maintaining these patches is not a big deal, really, but this is not the main
> point; the "need in real life" (1) is in fact the main one, and then, the "is
> this solution the best one ?" (2) the second one.
>
> About (1), as said in my first mail, as the namespaces and containers are being
> integrated into the mainline kernel, checkpoint/restart is (or will be) the next
> need.
> About (2), my solution propose to do that, as much as possible from userspace,
> to minimize the kernel impact. Of course, this is subject to discussion. My
> opinion is that doing a full checkpoint/restart from kernel space will need lot
> of new specific and intrusive code; I'm not sure that this will be acceptable by
> the community. But this is my opinion only. Discusion is opened.
>
>> Also such APIs allow creation of non-GPL checkpointing in user-space, which can be of concern as well.
>
> Honestly, I don't think this really a concern at all. I mean: I've never seen
> "this allows non-GPL binary and thus, this is bad" as an argument to reject a
> functionality, but I may be wrong, and thus, it can be discussed as well.
> I think the points (1) and (2) as stated above are the key ones.
>
> Pierre
>
>> Kirill
>>
>>
>> Pierre Peiffer wrote:
>>> Hi again,
>>>
>>> Thinking more about this, I think I must clarify why I choose this way.
>>> In fact, the idea of these patches is to provide the missing user APIs (or
>>> extend the existing ones) that allow to set or update _all_ properties of all
>>> IPCs, as needed in the case of the checkpoint/restart of an application (the
>>> current user API does not allow to specify an ID for a created IPC, for
>>> example). And this, without changing the existing API of course.
>>>

>>> And msgget(), semget() and shmget() does not have any parameter we can use to
>>> specify an ID.
>>> That's why I've decided to not change these routines and add a new control
>>> command, IP_SETID, with which we can can change the ID of an IPC. (that looks to
>>> me more straightforward and logical)
>>>
>>> Now, this patch is, in fact, only a preparation for the patch 10/15 which
>>> really complete the user API by adding this IPC_SETID command.
>>>
>>> (... continuing below ...)
>>>
>>> Alexey Dobriyan wrote:
>>>> On Tue, Jan 29, 2008 at 05:02:38PM +0100, pierre.peiffer@bull.net wrote:
>>>>> This patch provides three new API to change the ID of an existing
>>>>> System V IPCs.
>>>>>
>>>>> These APIs are:
>>>>> long msg_chid(struct ipc_namespace *ns, int id, int newid);
>>>>> long sem_chid(struct ipc_namespace *ns, int id, int newid);
>>>>> long shm_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>
>>>>> They return 0 or an error code in case of failure.
>>>>>
>>>>> They may be useful for setting a specific ID for an IPC when preparing
>>>>> a restart operation.
>>>>>
>>>>> To be successful, the following rules must be respected:
>>>>> - the IPC exists (of course...)
>>>>> - the new ID must satisfy the ID computation rule.
>>>>> - the entry in the idr corresponding to the new ID must be free.
>>>>> ipc/util.c | 48 ++
>>>>> ipc/util.h | 1 +
>>>>> 8 files changed, 197 insertions(+)
>>>> For the record, OpenVZ uses "create with predefined ID" method which
>>>> leads to less code. For example, change at the end is all we want from
>>>> ipc/util.c .
>>> And in fact, you do that from kernel space, you don't have the constraint to fit
>>> the existing user API.
>>> Again, this patch, even if it presents a new kernel API, is in fact a
>>> preparation for the next patch which introduces a new user API.
>>>
>>> Do you think that this could fit your need ?
>>>
>>
>

Containers mailing list
Containers@lists.linux-foundation.org

https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID
Posted by Cedric Le Goater on Thu, 31 Jan 2008 16:10:03 GMT
View Forum Message <> Reply to Message

Hello Kirill !

Kirill Korotaev wrote:
> Pierre,
>
> my point is that after you've added interface "set IPCID", you'll need more and more for checkpointing:
> - "create/setup conntrack" (otherwise connections get dropped),
> - "set task start time" (needed for Oracle checkpointing BTW),
> - "set some statistics counters (e.g. networking or taskstats)"
> - "restore inotify"
> and so on and so forth.

right. we know that we will have to handle a lot of these
and more and we will need an API for it :) so how should we
handle it ?

through a dedicated syscall that would be able to checkpoint
and/or restart a process, an ipc object, an ipc namespace, a
full container ? will it take a fd or a big binary blob ?

I personally really liked Pavel idea's of filesystem. but we
dropped the thread.

that's for the user API but we will need also kernel services
to expose (checkpoint) states and restore them. If it's too
early to talk about the user API, we could try first to refactor
the kernel internals to expose correctly what we need.

That's what Pierre's patchset is trying to do.

Cheers,

C.

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 14/15] (RFC) IPC/semaphores: prepare semundo code to work on another task
Posted by serue on Thu, 31 Jan 2008 18:01:25 GMT
View Forum Message <> Reply to Message

Quoting Pierre Peiffer (pierre.peiffer@bull.net):

```
>
>
> Serge E. Hallyn wrote:
> > Quoting pierre.peiffer@bull.net (pierre.peiffer@bull.net):
> >> From: Pierre Peiffer <pierre.peiffer@bull.net>
> >>
> >> In order to modify the semundo-list of a task from procfs, we must be able to
> >> work on any target task.
> >> But all the existing code playing with the semundo-list, currently works
> >> only on the 'current' task, and does not allow to specify any target task.
> >>
> >> This patch changes all these routines to allow them to work on a specified
> >> task, passed in parameter, instead of current.
> >>
> >> This is mainly a preparation for the semundo_write() operation, on the
> >> /proc/<pid>/semundo file, as provided in the next patch.
> >>
> >> Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
> >> ---
> >>
> >>  ipc/sem.c |   90 +++++++++++++++++++++++++++++++++++++++++++++---------------
> >>  1 file changed, 68 insertions(+), 22 deletions(-)
> >>
> >> Index: b/ipc/sem.c
> >> ===================================================================
> >> --- a/ipc/sem.c
> >> +++ b/ipc/sem.c
> >> @@ -1017,8 +1017,9 @@ asmlinkage long sys_semctl (int semid, i
> >>  }
> >>
> >>  /* If the task doesn't already have a undo_list, then allocate one
> >> - * here.  We guarantee there is only one thread using this undo list,
> >> - * and current is THE ONE
> >> + * here.
> >> + * The target task (tsk) is current in the general case, except when
> >> + * accessed from the procfs (ie when writting to /proc/<pid>/semundo)
> >>   *
> >>   * If this allocation and assignment succeeds, but later
> >>   * portions of this code fail, there is no need to free the sem_undo_list.
> >> @@ -1026,22 +1027,60 @@ asmlinkage long sys_semctl (int semid, i
> >>   * at exit time.
> >>   *
> >>   * This can block, so callers must hold no locks.
> >> + *
> >> + * Note: task_lock is used to synchronize 1. several possible concurrent
> >> + * creations and 2. the free of the undo_list (done when the task using it
> >> + * exits). In the second case, we check the PF_EXITING flag to not create
> >> + * an undo_list for a task which has exited.
> >> + * If there already is an undo_list for this task, there is no need
> >> + * to held the task-lock to retrieve it, as the pointer can not change
> >> + * afterwards.
> >>   */
> >> -static inline int get_undo_list(struct sem_undo_list **undo_listp)
> >> +static inline int get_undo_list(struct task_struct *tsk,
> >> +    struct sem_undo_list **ulp)
> >>  {
> >> - struct sem_undo_list *undo_list;
> >> + if (tsk->sysvsem.undo_list == NULL) {
> >> +  struct sem_undo_list *undo_list;
> >
> > Hmm, this is weird.  If there was no undo_list and
> > tsk!=current, you set the refcnt to 2.  But if there was an
> > undo list and tsk!=current, where do you inc the refcnt?
> >
>
> I inc it  outside this function, as I don't call get_undo_list() if there is an
> undo_list.
> This appears most clearly in the next patch, in semundo_open() for example.
```

Ok, so however unlikely, there is a flow that could cause you a problem:
T2 calls semundo_open() for T1. T1 does not yet have a semundolist.
T2.semundo_open() calls get_undo_list, just then T1 creats its own
semundo_list. T2 comes to top of get_undo_list() and see
tsk->sysvsem.undo_list != NULL, simply returns a pointer to the
undo_list. Now you never increment the count.

```
>
> >> - undo_list = current->sysvsem.undo_list;
> >> - if (!undo_list) {
> >> -  undo_list = kzalloc(sizeof(*undo_list), GFP_KERNEL);
> >> +  /* we must alloc a new one */
> >> +  undo_list = kmalloc(sizeof(*undo_list), GFP_KERNEL);
> >>    if (undo_list == NULL)
> >>     return -ENOMEM;
> >> +
> >> +  task_lock(tsk);
> >> +
> >> +  /* check again if there is an undo_list for this task */
> >> +  if (tsk->sysvsem.undo_list) {
> >> +   if (tsk != current)
> >> +    atomic_inc(&tsk->sysvsem.undo_list->refcnt);
> >> +   task_unlock(tsk);
> >> +   kfree(undo_list);
> >> +   goto out;
> >> +  }
> >> +
> >>    spin_lock_init(&undo_list->lock);
> >> -  atomic_set(&undo_list->refcnt, 1);
> >> -  undo_list->ns = get_ipc_ns(current->nsproxy->ipc_ns);
> >> -  current->sysvsem.undo_list = undo_list;
> >> +  /*
> >> +   * If tsk is not current (meaning that current is creating
> >> +   * a semundo_list for a target task through procfs), and if
> >> +   * it's not being exited then refcnt must be 2: the target
> >> +   * task tsk + current.
> >> +   */
> >> +  if (tsk == current)
> >> +   atomic_set(&undo_list->refcnt, 1);
> >> +  else if (!(tsk->flags & PF_EXITING))
> >> +   atomic_set(&undo_list->refcnt, 2);
> >> +  else {
> >> +   task_unlock(tsk);
> >> +   kfree(undo_list);
> >> +   return -EINVAL;
> >> +  }
> >> +  undo_list->ns = get_ipc_ns(tsk->nsproxy->ipc_ns);
> >> +  undo_list->proc_list = NULL;
> >> +  tsk->sysvsem.undo_list = undo_list;
> >> +  task_unlock(tsk);
> >>   }
> >> - *undo_listp = undo_list;
> >> +out:
> >> + *ulp = tsk->sysvsem.undo_list;
> >>   return 0;
> >>  }
> >>
> >> @@ -1065,17 +1104,12 @@ static struct sem_undo *lookup_undo(stru
> >>   return un;
> >>  }
> >>
> >> -static struct sem_undo *find_undo(struct ipc_namespace *ns, int semid)
> >> +static struct sem_undo *find_undo(struct sem_undo_list *ulp, int semid)
> >>  {
> >>   struct sem_array *sma;
> >> - struct sem_undo_list *ulp;
> >>   struct sem_undo *un, *new;
> >> + struct ipc_namespace *ns;
> >>   int nsems;
> >> - int error;
> >> -
> >> - error = get_undo_list(&ulp);
> >> - if (error)
> >> -  return ERR_PTR(error);
> >>
> >>   spin_lock(&ulp->lock);
> >>   un = lookup_undo(ulp, semid);
> >> @@ -1083,6 +1117,8 @@ static struct sem_undo *find_undo(struct
> >>   if (likely(un!=NULL))
> >>    goto out;
> >>
> >> + ns = ulp->ns;
> >> +
> >>   /* no undo structure around - allocate one. */
> >>   sma = sem_lock_check(ns, semid);
> >>   if (IS_ERR(sma))
> >> @@ -1133,6 +1169,7 @@ asmlinkage long sys_semtimedop(int semid
> >>   struct sem_array *sma;
> >>   struct sembuf fast_sops[SEMOPM_FAST];
> >>   struct sembuf* sops = fast_sops, *sop;
> >> + struct sem_undo_list *ulp;
> >>   struct sem_undo *un;
> >>   int undos = 0, alter = 0, max;
> >>   struct sem_queue queue;
> >> @@ -1177,9 +1214,13 @@ asmlinkage long sys_semtimedop(int semid
> >>     alter = 1;
> >>   }
> >>
> >> + error = get_undo_list(current, &ulp);
> >> + if (error)
> >> +  goto out_free;
> >> +
> >>  retry_undos:
> >>   if (undos) {
> >> -  un = find_undo(ns, semid);
> >> +  un = find_undo(ulp, semid);
> >>    if (IS_ERR(un)) {
> >>     error = PTR_ERR(un);
> >>     goto out_free;
> >> @@ -1305,7 +1346,7 @@ int copy_semundo(unsigned long clone_fla
> >>   int error;
> >>
> >>   if (clone_flags & CLONE_SYSVSEM) {
> >> -  error = get_undo_list(&undo_list);
> >> +  error = get_undo_list(current, &undo_list);
> >>    if (error)
> >>     return error;
> >>    atomic_inc(&undo_list->refcnt);
> >> @@ -1405,10 +1446,15 @@ next_entry:
> >>   kfree(undo_list);
> >>  }
> >>
> >> -/* called from do_exit() */
> >> +/* exit_sem: called from do_exit()
> >> + * task_lock is used to synchronize with get_undo_list()
> >
> > Ok I had to think about this again.  I'd like the comment
> > here to point out that the task_lock here acts as a barrier
> > between the prior setting of PF_EXITING and the undo_list
> > being freed here, so that get_undo_list() will either see
> > PF_EXITING is NOT in the tsk->flags, in which case it will
> > insert the undo_list before the task_lock() is grabbed here,
> > and with count=2, so that it gets correctly put here in
> > exit_sem, or it will see PF_EXITING set and cancel the
> > undo_list it was creating.
> >
>
> Yep, I will add this to clarify this point.
>
> Thanks Serge.
>
> P.
>
> >> + */
> >>  void exit_sem(struct task_struct *tsk)
> >>  {
> >> - struct sem_undo_list *ul = tsk->sysvsem.undo_list;
> >> + struct sem_undo_list *ul;
> >> + task_lock(tsk);
> >> + ul = tsk->sysvsem.undo_list;
> >> + task_unlock(tsk);
> >>   if (ul) {
> >>    rcu_assign_pointer(tsk->sysvsem.undo_list, NULL);
> >>    synchronize_rcu();
> >>
> >> --
> >> Pierre Peiffer
> >> _______________________________________________
> >> Containers mailing list
> >> Containers@lists.linux-foundation.org
> >> https://lists.linux-foundation.org/mailman/listinfo/containers
> >
> >
>
> --
> Pierre Peiffer
```

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 14/15] (RFC) IPC/semaphores: prepare semundo code to work on another task
Posted by Pierre Peiffer on Fri, 01 Feb 2008 12:09:48 GMT

View Forum Message <> Reply to Message

```
Serge E. Hallyn wrote:
> Quoting Pierre Peiffer (pierre.peiffer@bull.net):
>>
>> Serge E. Hallyn wrote:
>>> Quoting pierre.peiffer@bull.net (pierre.peiffer@bull.net):
>>>> From: Pierre Peiffer <pierre.peiffer@bull.net>
>>>>
>>>> In order to modify the semundo-list of a task from procfs, we must be able to
>>>> work on any target task.
>>>> But all the existing code playing with the semundo-list, currently works
>>>> only on the 'current' task, and does not allow to specify any target task.
>>>>
>>>> This patch changes all these routines to allow them to work on a specified
>>>> task, passed in parameter, instead of current.
>>>>
>>>> This is mainly a preparation for the semundo_write() operation, on the
>>>> /proc/<pid>/semundo file, as provided in the next patch.
>>>>
>>>> Signed-off-by: Pierre Peiffer <pierre.peiffer@bull.net>
>>>> ---
>>>>
>>>>  ipc/sem.c |   90 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++----------------------
>>>>  1 file changed, 68 insertions(+), 22 deletions(-)
>>>>
>>>> Index: b/ipc/sem.c
>>>> ===================================================================
>>>> --- a/ipc/sem.c
>>>> +++ b/ipc/sem.c
>>>> @@ -1017,8 +1017,9 @@ asmlinkage long sys_semctl (int semid, i
>>>>  }
>>>>
>>>>  /* If the task doesn't already have a undo_list, then allocate one
>>>> - * here.  We guarantee there is only one thread using this undo list,
>>>> - * and current is THE ONE
>>>> + * here.
>>>> + * The target task (tsk) is current in the general case, except when
>>>> + * accessed from the procfs (ie when writting to /proc/<pid>/semundo)
```

```
>>>>   *
>>>>   * If this allocation and assignment succeeds, but later
>>>>   * portions of this code fail, there is no need to free the sem_undo_list.
>>>> @@ -1026,22 +1027,60 @@ asmlinkage long sys_semctl (int semid, i
>>>>   * at exit time.
>>>>   *
>>>>   * This can block, so callers must hold no locks.
>>>> + *
>>>> + * Note: task_lock is used to synchronize 1. several possible concurrent
>>>> + * creations and 2. the free of the undo_list (done when the task using it
>>>> + * exits). In the second case, we check the PF_EXITING flag to not create
>>>> + * an undo_list for a task which has exited.
>>>> + * If there already is an undo_list for this task, there is no need
>>>> + * to held the task-lock to retrieve it, as the pointer can not change
>>>> + * afterwards.
>>>>   */
>>>> -static inline int get_undo_list(struct sem_undo_list **undo_listp)
>>>> +static inline int get_undo_list(struct task_struct *tsk,
>>>> +    struct sem_undo_list **ulp)
>>>>  {
>>>> - struct sem_undo_list *undo_list;
>>>> + if (tsk->sysvsem.undo_list == NULL) {
>>>> +  struct sem_undo_list *undo_list;
>>> Hmm, this is weird.  If there was no undo_list and
>>> tsk!=current, you set the refcnt to 2.  But if there was an
>>> undo list and tsk!=current, where do you inc the refcnt?
>>>
>> I inc it  outside this function, as I don't call get_undo_list() if there is an
>> undo_list.
>> This appears most clearly in the next patch, in semundo_open() for example.
>
> Ok, so however unlikely, there is a flow that could cause you a problem:
> T2 calls semundo_open() for T1.  T1 does not yet have a semundolist.
> T2.semundo_open() calls get_undo_list, just then T1 creats its own
> semundo_list.  T2 comes to top of get_undo_list() and see
> tsk->sysvsem.undo_list != NULL, simply returns a pointer to the
> undo_list.  Now you never increment the count.
>
```

Right.

And yesterday, with more testing in the corners, I've found another issue: if I use /proc/self/semundo, I don't have tsk != current and the refcnt is wrong too.

Thanks for finding this !

P.

```
>>>> - undo_list = current->sysvsem.undo_list;
```

```
>>>> - if (!undo_list) {
>>>> -  undo_list = kzalloc(sizeof(*undo_list), GFP_KERNEL);
>>>> +  /* we must alloc a new one */
>>>> +  undo_list = kmalloc(sizeof(*undo_list), GFP_KERNEL);
>>>>    if (undo_list == NULL)
>>>>     return -ENOMEM;
>>>> +
>>>> +  task_lock(tsk);
>>>> +
>>>> +  /* check again if there is an undo_list for this task */
>>>> +  if (tsk->sysvsem.undo_list) {
>>>> +   if (tsk != current)
>>>> +    atomic_inc(&tsk->sysvsem.undo_list->refcnt);
>>>> +   task_unlock(tsk);
>>>> +   kfree(undo_list);
>>>> +   goto out;
>>>> +  }
>>>> +
>>>>    spin_lock_init(&undo_list->lock);
>>>> -  atomic_set(&undo_list->refcnt, 1);
>>>> -  undo_list->ns = get_ipc_ns(current->nsproxy->ipc_ns);
>>>> -  current->sysvsem.undo_list = undo_list;
>>>> +  /*
>>>> +   * If tsk is not current (meaning that current is creating
>>>> +   * a semundo_list for a target task through procfs), and if
>>>> +   * it's not being exited then refcnt must be 2: the target
>>>> +   * task tsk + current.
>>>> +   */
>>>> +  if (tsk == current)
>>>> +   atomic_set(&undo_list->refcnt, 1);
>>>> +  else if (!(tsk->flags & PF_EXITING))
>>>> +   atomic_set(&undo_list->refcnt, 2);
>>>> +  else {
>>>> +   task_unlock(tsk);
>>>> +   kfree(undo_list);
>>>> +   return -EINVAL;
>>>> +  }
>>>> +  undo_list->ns = get_ipc_ns(tsk->nsproxy->ipc_ns);
>>>> +  undo_list->proc_list = NULL;
>>>> +  tsk->sysvsem.undo_list = undo_list;
>>>> +  task_unlock(tsk);
>>>>   }
>>>> - *undo_listp = undo_list;
>>>> +out:
>>>> + *ulp = tsk->sysvsem.undo_list;
>>>>   return 0;
>>>>  }
>>>>
```

```
>>>> @@ -1065,17 +1104,12 @@ static struct sem_undo *lookup_undo(stru
>>>>   return un;
>>>>  }
>>>>
>>>> -static struct sem_undo *find_undo(struct ipc_namespace *ns, int semid)
>>>> +static struct sem_undo *find_undo(struct sem_undo_list *ulp, int semid)
>>>>  {
>>>>   struct sem_array *sma;
>>>> - struct sem_undo_list *ulp;
>>>>   struct sem_undo *un, *new;
>>>> + struct ipc_namespace *ns;
>>>>   int nsems;
>>>> - int error;
>>>> -
>>>> - error = get_undo_list(&ulp);
>>>> - if (error)
>>>> -  return ERR_PTR(error);
>>>>
>>>>   spin_lock(&ulp->lock);
>>>>   un = lookup_undo(ulp, semid);
>>>> @@ -1083,6 +1117,8 @@ static struct sem_undo *find_undo(struct
>>>>   if (likely(un!=NULL))
>>>>    goto out;
>>>>
>>>> + ns = ulp->ns;
>>>> +
>>>>   /* no undo structure around - allocate one. */
>>>>   sma = sem_lock_check(ns, semid);
>>>>   if (IS_ERR(sma))
>>>> @@ -1133,6 +1169,7 @@ asmlinkage long sys_semtimedop(int semid
>>>>   struct sem_array *sma;
>>>>   struct sembuf fast_sops[SEMOPM_FAST];
>>>>   struct sembuf* sops = fast_sops, *sop;
>>>> + struct sem_undo_list *ulp;
>>>>   struct sem_undo *un;
>>>>   int undos = 0, alter = 0, max;
>>>>   struct sem_queue queue;
>>>> @@ -1177,9 +1214,13 @@ asmlinkage long sys_semtimedop(int semid
>>>>     alter = 1;
>>>>   }
>>>>
>>>> + error = get_undo_list(current, &ulp);
>>>> + if (error)
>>>> +  goto out_free;
>>>> +
>>>>  retry_undos:
>>>>   if (undos) {
>>>> -  un = find_undo(ns, semid);
```

```
>>>> +  un = find_undo(ulp, semid);
>>>>    if (IS_ERR(un)) {
>>>>     error = PTR_ERR(un);
>>>>     goto out_free;
>>>> @@ -1305,7 +1346,7 @@ int copy_semundo(unsigned long clone_fla
>>>>   int error;
>>>>
>>>>   if (clone_flags & CLONE_SYSVSEM) {
>>>> -  error = get_undo_list(&undo_list);
>>>> +  error = get_undo_list(current, &undo_list);
>>>>    if (error)
>>>>     return error;
>>>>    atomic_inc(&undo_list->refcnt);
>>>> @@ -1405,10 +1446,15 @@ next_entry:
>>>>   kfree(undo_list);
>>>>  }
>>>>
>>>> -/* called from do_exit() */
>>>> +/* exit_sem: called from do_exit()
>>>> + * task_lock is used to synchronize with get_undo_list()
>>> Ok I had to think about this again.  I'd like the comment
>>> here to point out that the task_lock here acts as a barrier
>>> between the prior setting of PF_EXITING and the undo_list
>>> being freed here, so that get_undo_list() will either see
>>> PF_EXITING is NOT in the tsk->flags, in which case it will
>>> insert the undo_list before the task_lock() is grabbed here,
>>> and with count=2, so that it gets correctly put here in
>>> exit_sem, or it will see PF_EXITING set and cancel the
>>> undo_list it was creating.
>>>
>> Yep, I will add this to clarify this point.
>>
>> Thanks Serge.
>>
>> P.
>>
>>>> + */
>>>>  void exit_sem(struct task_struct *tsk)
>>>>  {
>>>> - struct sem_undo_list *ul = tsk->sysvsem.undo_list;
>>>> + struct sem_undo_list *ul;
>>>> + task_lock(tsk);
>>>> + ul = tsk->sysvsem.undo_list;
>>>> + task_unlock(tsk);
>>>>   if (ul) {
>>>>    rcu_assign_pointer(tsk->sysvsem.undo_list, NULL);
>>>>    synchronize_rcu();
>>>>
```

>>>> --
>>>> Pierre Peiffer
>>>> ____
>>>> Containers mailing list
>>>> Containers@lists.linux-foundation.org
>>>> https://lists.linux-foundation.org/mailman/listinfo/containers
>>>
>> --
>> Pierre Peiffer
>
>

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 2.6.24-rc8-mm1 00/15] IPC: code rewrite + new functionalities
Posted by Pavel Machek on Sat, 02 Feb 2008 18:23:52 GMT

View Forum Message <> Reply to Message

Hi!

> * Patches 9 to 15 propose to add some functionalities, and thus are
> submitted here for RFC, about both the interest and their implementation.
> These functionalities are:
>   - Two new control-commands:
> . IPC_SETID: to change an IPC's id.
> . IPC_SETALL: behaves as IPC_SET, except that it also sets all time
>   and pid values)
>   - add a /proc/<pid>/semundo file to read and write the undo values of
> some semaphores for a given process.
>
> As the namespaces and the "containers" are being integrated in the
> kernel, these functionalities may be a first step to implement the
> checkpoint/restart of an application: in fact the existing API does not allow
> to specify or to change an ID when creating an IPC, when restarting an
> application, and the times/pids values of each IPCs are also altered. May be
> someone may find another interest about this ?
>
> So again, comments are welcome.

Checkpoint/restart is nice, but... sysV ipc is broken by design, do we
really want to extend it?
	Pavel

--
(english) http://www.livejournal.com/~pavelmachek
(cesky, pictures) http://atrey.karlin.mff.cuni.cz/~pavel/picture/horses/blog.html

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID
Posted by dev on Mon, 04 Feb 2008 13:41:26 GMT

View Forum Message <> Reply to Message

Cedric Le Goater wrote:
> Hello Kirill !
>
> Kirill Korotaev wrote:
>> Pierre,
>>
>> my point is that after you've added interface "set IPCID", you'll need
>> more and more for checkpointing:
>> - "create/setup conntrack" (otherwise connections get dropped),
>> - "set task start time" (needed for Oracle checkpointing BTW),
>> - "set some statistics counters (e.g. networking or taskstats)"
>> - "restore inotify"
>> and so on and so forth.
>
> right. we know that we will have to handle a lot of these
> and more and we will need an API for it :) so how should we handle it ?
> through a dedicated syscall that would be able to checkpoint and/or
> restart a process, an ipc object, an ipc namespace, a full container ?
> will it take a fd or a big binary blob ?
> I personally really liked Pavel idea's of filesystem. but we dropped the
> thread.

Imho having a file system interface means having all its problems.
Imagine you have some information about tasks exported with a file system interface.
Obviously to collect the information you have to hold some spinlock like tasklist_lock or similar.
Obviously, you have to drop the lock between sys_read() syscalls.
So interface gets much more complicated - you have to rescan the objects and somehow find the place where
you stopped previous read. Or you have to to force reader to read everything at once.

> that's for the user API but we will need also kernel services to expose
> (checkpoint) states and restore them. If it's too
> early to talk about the user API, we could try first to refactor
> the kernel internals to expose correctly what we need.

That's what I would start with.

> That's what Pierre's patchset is trying to do.

Not exactly. For checkpointing/restoring we actually need only one new API call for each subsystem - create some object with given ID (and maybe parameters, if they are not dynamically changeable by user).
While Pierre's patchset adds different API call - change object ID.

Thanks,
Kirill

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 2.6.24-rc8-mm1 00/15] IPC: code rewrite + new functionalities
Posted by Pierre Peiffer on Mon, 04 Feb 2008 13:52:17 GMT
View Forum Message <> Reply to Message

Pavel Machek wrote:
> Hi!
>
>> * Patches 9 to 15 propose to add some functionalities, and thus are
>> submitted here for RFC, about both the interest and their implementation.
>> These functionalities are:
>>   - Two new control-commands:
>> . IPC_SETID: to change an IPC's id.
>> . IPC_SETALL: behaves as IPC_SET, except that it also sets all time
>>   and pid values)
>>   - add a /proc/<pid>/semundo file to read and write the undo values of
>> some semaphores for a given process.
>>
>> As the namespaces and the "containers" are being integrated in the
>> kernel, these functionalities may be a first step to implement the
>> checkpoint/restart of an application: in fact the existing API does not allow
>> to specify or to change an ID when creating an IPC, when restarting an
>> application, and the times/pids values of each IPCs are also altered. May be
>> someone may find another interest about this ?
>>
>> So again, comments are welcome.
>
> Checkpoint/restart is nice, but... sysV ipc is broken by design, do we
> really want to extend it?

If we want to support all kind of applications, yes, we must also support

SysVipc. We must support all kernel subsystems at the end.
I've started with IPC, because it's relatively simple and isolated.

--
Pierre

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID
Posted by Pavel Emelianov on Mon, 04 Feb 2008 14:06:23 GMT

View Forum Message <> Reply to Message

Kirill Korotaev wrote:
>
> Cedric Le Goater wrote:
>> Hello Kirill !
>>
>> Kirill Korotaev wrote:
>>> Pierre,
>>>
>>> my point is that after you've added interface "set IPCID", you'll need
>>> more and more for checkpointing:
>>> - "create/setup conntrack" (otherwise connections get dropped),
>>> - "set task start time" (needed for Oracle checkpointing BTW),
>>> - "set some statistics counters (e.g. networking or taskstats)"
>>> - "restore inotify"
>>> and so on and so forth.
>> right. we know that we will have to handle a lot of these
>> and more and we will need an API for it :) so how should we handle it ?
>> through a dedicated syscall that would be able to checkpoint and/or
>> restart a process, an ipc object, an ipc namespace, a full container ?
>> will it take a fd or a big binary blob ?
>> I personally really liked Pavel idea's of filesystem. but we dropped the
>> thread.
>
> Imho having a file system interface means having all its problems.
> Imagine you have some information about tasks exported with a file system interface.
> Obviously to collect the information you have to hold some spinlock like tasklist_lock or similar.
> Obviously, you have to drop the lock between sys_read() syscalls.
> So interface gets much more complicated - you have to rescan the objects and somehow find the place where
> you stopped previous read. Or you have to to force reader to read everything at once.

To remember the place when we stopped previous read we have a "pos" counter
on the struct file.

Actually, tar utility, that I propose to perform the most simple migration
reads the directory contents with 4Kb buffer - that's enough for ~500 tasks.

Besides, is this a real problem for a frozen container?

>> that's for the user API but we will need also kernel services to expose
>> (checkpoint) states and restore them. If it's too
>> early to talk about the user API, we could try first to refactor
>> the kernel internals to expose correctly what we need.
>
> That's what I would start with.
>
>> That's what Pierre's patchset is trying to do.
>
> Not exactly. For checkpointing/restoring we actually need only one new API call for each
> subsystem - create some object with given ID (and maybe parameters, if they are not
dynamically changeable by user).
> While Pierre's patchset adds different API call - change object ID.
>
> Thanks,
> Kirill
> ____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers
>

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID
Posted by Daniel Lezcano on Mon, 04 Feb 2008 15:00:33 GMT

View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> Kirill Korotaev wrote:
>> Cedric Le Goater wrote:
>>> Hello Kirill !
>>>
>>> Kirill Korotaev wrote:
>>>> Pierre,
>>>>

>>>> my point is that after you've added interface "set IPCID", you'll need
>>>> more and more for checkpointing:
>>>> - "create/setup conntrack" (otherwise connections get dropped),
>>>> - "set task start time" (needed for Oracle checkpointing BTW),
>>>> - "set some statistics counters (e.g. networking or taskstats)"
>>>> - "restore inotify"
>>>> and so on and so forth.
>>> right. we know that we will have to handle a lot of these
>>> and more and we will need an API for it :) so how should we handle it ?
>>> through a dedicated syscall that would be able to checkpoint and/or
>>> restart a process, an ipc object, an ipc namespace, a full container ?
>>> will it take a fd or a big binary blob ?
>>> I personally really liked Pavel idea's of filesystem. but we dropped the
>>> thread.
>> Imho having a file system interface means having all its problems.
>> Imagine you have some information about tasks exported with a file system interface.
>> Obviously to collect the information you have to hold some spinlock like tasklist_lock or similar.
>> Obviously, you have to drop the lock between sys_read() syscalls.
>> So interface gets much more complicated - you have to rescan the objects and somehow find the place where
>> you stopped previous read. Or you have to to force reader to read everything at once.
>
> To remember the place when we stopped previous read we have a "pos" counter
> on the struct file.
>
> Actually, tar utility, that I propose to perform the most simple migration
> reads the directory contents with 4Kb buffer - that's enough for ~500 tasks.
>
> Besides, is this a real problem for a frozen container?

I like the idea of a C/R filesystem. Does it implies a specific user
space program to orchestrate the checkpoint/restart of the different
subsystems ? I mean the checkpoint is easy but what about the restart ?
We must ensure, for example to restore a process before restoring the fd
associated to it, or restore a deleted file before restoring the fd
opened to it, no ?

Subject: Re: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID
Posted by Pavel Emelianov on Mon, 04 Feb 2008 15:16:43 GMT
View Forum Message <> Reply to Message

Daniel Lezcano wrote:
> Pavel Emelyanov wrote:
>> Kirill Korotaev wrote:
>>> Cedric Le Goater wrote:
>>>> Hello Kirill !
>>>>
>>>> Kirill Korotaev wrote:
>>>>> Pierre,
>>>>>
>>>>> my point is that after you've added interface "set IPCID", you'll need
>>>>> more and more for checkpointing:
>>>>> - "create/setup conntrack" (otherwise connections get dropped),
>>>>> - "set task start time" (needed for Oracle checkpointing BTW),
>>>>> - "set some statistics counters (e.g. networking or taskstats)"
>>>>> - "restore inotify"
>>>>> and so on and so forth.
>>>> right. we know that we will have to handle a lot of these
>>>> and more and we will need an API for it :) so how should we handle it ?
>>>> through a dedicated syscall that would be able to checkpoint and/or
>>>> restart a process, an ipc object, an ipc namespace, a full container ?
>>>> will it take a fd or a big binary blob ?
>>>> I personally really liked Pavel idea's of filesystem. but we dropped the
>>>> thread.
>>> Imho having a file system interface means having all its problems.
>>> Imagine you have some information about tasks exported with a file system interface.
>>> Obviously to collect the information you have to hold some spinlock like tasklist_lock or similar.
>>> Obviously, you have to drop the lock between sys_read() syscalls.
>>> So interface gets much more complicated - you have to rescan the objects and somehow find the place where
>>> you stopped previous read. Or you have to to force reader to read everything at once.
>> To remember the place when we stopped previous read we have a "pos" counter
>> on the struct file.
>>
>> Actually, tar utility, that I propose to perform the most simple migration
>> reads the directory contents with 4Kb buffer - that's enough for ~500 tasks.
>>
>> Besides, is this a real problem for a frozen container?
>
> I like the idea of a C/R filesystem. Does it implies a specific user
> space program to orchestrate the checkpoint/restart of the different
> subsystems ? I mean the checkpoint is easy but what about the restart ?

I though about smth like "writing to this fs causes restore process".

> We must ensure, for example to restore a process before restoring the fd
> associated to it, or restore a deleted file before restoring the fd

This is achieved by tar automatically - it extracts files in the order
of archiving. Thus is we provide them in correct order we'll get them
in correct one as well.

> opened to it, no ?
>
>
>
>
>
>

Subject: Re: [PATCH 2.6.24-rc8-mm1 00/15] IPC: code rewrite + new functionalities
Posted by Benjamin Thery on Mon, 04 Feb 2008 15:44:34 GMT

View Forum Message <> Reply to Message

Pavel Machek wrote:
> Hi!
>
>> * Patches 9 to 15 propose to add some functionalities, and thus are
>> submitted here for RFC, about both the interest and their implementation.
>> These functionalities are:
>>   - Two new control-commands:
>> . IPC_SETID: to change an IPC's id.
>> . IPC_SETALL: behaves as IPC_SET, except that it also sets all time
>>   and pid values)
>>   - add a /proc/<pid>/semundo file to read and write the undo values of
>> some semaphores for a given process.
>>
>> As the namespaces and the "containers" are being integrated in the
>> kernel, these functionalities may be a first step to implement the
>> checkpoint/restart of an application: in fact the existing API does not allow
>> to specify or to change an ID when creating an IPC, when restarting an
>> application, and the times/pids values of each IPCs are also altered. May be
>> someone may find another interest about this ?
>>
>> So again, comments are welcome.
>

> Checkpoint/restart is nice, but... sysV ipc is broken by design, do we
> really want to extend it?
>   Pavel

For my personal culture, what do you mean by "broken by design"?

Even if it's broken, don't you think some people could be interested in checkpointing "legacy" applications that use SysV IPC?

Benjamin

--
Benjamin Thery - BULL/DT/Open Software R&D

   http://www.bull.com

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 2.6.24-rc8-mm1 00/15] IPC: code rewrite + new functionalities
Posted by Pavel Machek on Mon, 04 Feb 2008 19:51:50 GMT
View Forum Message <> Reply to Message

Hi!

>>> As the namespaces and the "containers" are being integrated in the
>>> kernel, these functionalities may be a first step to implement the
>>> checkpoint/restart of an application: in fact the existing API does not allow
>>> to specify or to change an ID when creating an IPC, when restarting an
>>> application, and the times/pids values of each IPCs are also altered. May be
>>> someone may find another interest about this ?
>>>
>>> So again, comments are welcome.
>>
>> Checkpoint/restart is nice, but... sysV ipc is broken by design, do we
>> really want to extend it?

>
> For my personal culture, what do you mean by "broken by design"?

```
       int shmget(key_t key, size_t size, int shmflg);
```

....so how do you produce key in a way that is guaranteed not to
interfere with other uses?
        Pavel

--
(english) http://www.livejournal.com/~pavelmachek
(cesky, pictures) http://atrey.karlin.mff.cuni.cz/~pavel/picture/horses/blog.html

Subject: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID
Posted by Oren Laadan on Tue, 05 Feb 2008 09:51:09 GMT

View Forum Message <> Reply to Message

I strongly second Kirill on this matter.

IMHO, we should _avoid_ as much as possible exposing internal kernel state to applications, unless a _real_ need for it is _clearly_ demonstrated. The reasons for this are quite obvious.

It isn't strictly necessary to export a new interface in order to support checkpoint/restart. **. Hence, I think that the speculation "we may need it in the future" is too abstract and isn't a good excuse to commit to a new, currently unneeded, interface. Should the need arise in the future, it will be easy to design a new interface (also based on aggregated experience until then).

** In fact, the suggested interface may prove problematic (as noted earlier in this thread): if you first create the resource with some arbitrary identifier and then modify the identifier (in our case, IPC id), then the restart procedure is bound to execute sequentially, because of lack of atomicity.

That said, I suggest the following method instead (this is the method we use in Zap to determine the desired resource identifier when a new resource is allocated; I recall that we had discussed it in the past, perhaps the mini-summit in september ?):

1) The process/thread tells the kernel that it wishes to pre-determine the resource identifier of a subsequent call (this can be done via a new syscall, or by writing to /proc/self/...).

2) Each system call that allocates a resource and assigns an identifier is modified to check this per-thread field first; if it is set then it will attempt to allocate that particular value (if already taken, return an error, eg. EBUSY). Otherwise it will proceed as it is today.

(I left out some details - eg. the kernel will keep the desire value

on a per-thread field, when it will be reset, whether we want to also
tag the field with its type and so on, but the idea is now clear).

The main two advantages are that first, we don't need to devise a new
method for every syscall that allocates said resources (sigh... just
think of clone() nightmare to add a new argument); second, the change
is incremental: first code the mechanism to set the field, then add
support in the IPC subsystem, later in the DEVPTS, then in clone and
so forth.

Oren.

Pierre Peiffer wrote:
>
> Kirill Korotaev wrote:
>> Why user space can need this API? for checkpointing only?
>
> I would say "at least for checkpointing"... ;) May be someone else may find an
> interest about this for something else.
> In fact, I'm sure that you have some interest in checkpointing; and thus, you
> have probably some ideas in mind; but whatever the solution you will propose,
> I'm pretty sure that I could say the same thing for your solution.
> And what I finally think is: even if it's for "checkpointing only", if many
> people are interested by this, it may be sufficient to push this ?
>
>> Then I would not consider it for inclusion until it is clear how to implement checkpointing.
>> As for me personally - I'm against exporting such APIs, since they are not needed in real-life user space applications and maintaining it forever for compatibility doesn't worth it.
>
> Maintaining these patches is not a big deal, really, but this is not the main
> point; the "need in real life" (1) is in fact the main one, and then, the "is
> this solution the best one ?" (2) the second one.
>
> About (1), as said in my first mail, as the namespaces and containers are being
> integrated into the mainline kernel, checkpoint/restart is (or will be) the next
> need.
> About (2), my solution propose to do that, as much as possible from userspace,
> to minimize the kernel impact. Of course, this is subject to discussion. My
> opinion is that doing a full checkpoint/restart from kernel space will need lot
> of new specific and intrusive code; I'm not sure that this will be acceptable by
> the community. But this is my opinion only. Discusion is opened.
>
>> Also such APIs allow creation of non-GPL checkpointing in user-space, which can be of concern as well.
>
> Honestly, I don't think this really a concern at all. I mean: I've never seen
> "this allows non-GPL binary and thus, this is bad" as an argument to reject a
> functionality, but I may be wrong, and thus, it can be discussed as well.

> I think the points (1) and (2) as stated above are the key ones.
>
> Pierre
>
>> Kirill
>>
>>
>> Pierre Peiffer wrote:
>>> Hi again,
>>>
>>> Thinking more about this, I think I must clarify why I choose this way.
>>> In fact, the idea of these patches is to provide the missing user APIs (or
>>> extend the existing ones) that allow to set or update _all_ properties of all
>>> IPCs, as needed in the case of the checkpoint/restart of an application (the
>>> current user API does not allow to specify an ID for a created IPC, for
>>> example). And this, without changing the existing API of course.
>>>
>>> And msgget(), semget() and shmget() does not have any parameter we can use to
>>> specify an ID.
>>> That's why I've decided to not change these routines and add a new control
>>> command, IP_SETID, with which we can can change the ID of an IPC. (that looks to
>>> me more straightforward and logical)
>>>
>>> Now, this patch is, in fact, only a preparation for the patch 10/15 which
>>> really complete the user API by adding this IPC_SETID command.
>>>
>>> (... continuing below ...)
>>>
>>> Alexey Dobriyan wrote:
>>>> On Tue, Jan 29, 2008 at 05:02:38PM +0100, pierre.peiffer@bull.net wrote:
>>>>> This patch provides three new API to change the ID of an existing
>>>>> System V IPCs.
>>>>>
>>>>> These APIs are:
>>>>> long msg_chid(struct ipc_namespace *ns, int id, int newid);
>>>>> long sem_chid(struct ipc_namespace *ns, int id, int newid);
>>>>> long shm_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>
>>>>> They return 0 or an error code in case of failure.
>>>>>
>>>>> They may be useful for setting a specific ID for an IPC when preparing
>>>>> a restart operation.
>>>>>
>>>>> To be successful, the following rules must be respected:
>>>>> - the IPC exists (of course...)
>>>>> - the new ID must satisfy the ID computation rule.
>>>>> - the entry in the idr corresponding to the new ID must be free.
>>>>> ipc/util.c | 48 ++

>>>>> ipc/util.h | 1 +
>>>>> 8 files changed, 197 insertions(+)
>>>> For the record, OpenVZ uses "create with predefined ID" method which
>>>> leads to less code. For example, change at the end is all we want from
>>>> ipc/util.c .
>>> And in fact, you do that from kernel space, you don't have the constraint to fit
>>> the existing user API.
>>> Again, this patch, even if it presents a new kernel API, is in fact a
>>> preparation for the next patch which introduces a new user API.
>>>
>>> Do you think that this could fit your need ?
>>>
>>
>

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID

Posted by Dave Hansen on Tue, 05 Feb 2008 18:00:25 GMT

View Forum Message <> Reply to Message

On Tue, 2008-02-05 at 04:51 -0500, Oren Laadan wrote:
> That said, I suggest the following method instead (this is the method
> we use in Zap to determine the desired resource identifier when a new
> resource is allocated; I recall that we had discussed it in the past,
> perhaps the mini-summit in september ?):
>
> 1) The process/thread tells the kernel that it wishes to pre-determine
> the resource identifier of a subsequent call (this can be done via a
> new syscall, or by writing to /proc/self/...).
>
> 2) Each system call that allocates a resource and assigns an
> identifier
> is modified to check this per-thread field first; if it is set then
> it will attempt to allocate that particular value (if already taken,
> return an error, eg. EBUSY). Otherwise it will proceed as it is today.

You forgot to attach the patch to your mail. ;)

-- Dave

____
Containers mailing list
Containers@lists.linux-foundation.org

https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID
Posted by serue on Tue, 05 Feb 2008 18:42:34 GMT

View Forum Message <> Reply to Message

Quoting Oren Laadan (orenl@cs.columbia.edu):

>
> I strongly second Kirill on this matter.
>
> IMHO, we should _avoid_ as much as possible exposing internal kernel
> state to applications, unless a _real_ need for it is _clearly_
> demonstrated. The reasons for this are quite obvious.

Hmm, sure, but this sentence is designed to make us want to agree. Yes, we want to avoid exporting kernel internals, but generally that means things like the precise layout of the task_struct. What Pierre is doing is in fact the opposite, exporting resource information in a kernel version invariant way.

In fact, the very reason not to go the route you and Pavel are advocating is that if we just dump task state to a file or filesystem from the kernel in one shot, we'll be much more tempted to lay out data in a way that exports and ends up depending on kernel internals. So we'll just want to read and write the task_struct verbatim.

So, there are two very different approaches we can start with. Whichever one we follow, we want to avoid having kernel version dependencies. They both have their merits to be sure.

But note that in either case we need to deal with a bunch of locking. So getting back to Pierre's patchset, IIRC 1-8 are cleanups worth doing no matter 1. 9-11 sound like they are contentuous until we decide whether we want to go with a create_with_id() type approach or a set_id(). 12 is IMO a good locking cleanup regardless. 13 and 15 are contentous until we decide whether we want userspace-controlled checkpoint or a one-shot fs. 14 IMO is useful for both c/r approaches.

Is that pretty accurate?

> It isn't strictly necessary to export a new interface in order to
> support checkpoint/restart. **. Hence, I think that the speculation
> "we may need it in the future" is too abstract and isn't a good
> excuse to commit to a new, currently unneeded, interface.

OTOH it did succeed in starting some conversation :)

> Should the
> need arise in the future, it will be easy to design a new interface
> (also based on aggregated experience until then).

What aggregated experience? We have to start somewhere...

> ** In fact, the suggested interface may prove problematic (as noted
> earlier in this thread): if you first create the resource with some
> arbitrary identifier and then modify the identifier (in our case,
> IPC id), then the restart procedure is bound to execute sequentially,
> because of lack of atomicity.

Hmm? Lack of atomicity wrt what? All the tasks being restarted were checkpointed at the same time so there will be no conflict in the requested IDs, so I don't know what you're referring to.

> That said, I suggest the following method instead (this is the method
> we use in Zap to determine the desired resource identifier when a new
> resource is allocated; I recall that we had discussed it in the past,
> perhaps the mini-summit in september ?):
>
> 1) The process/thread tells the kernel that it wishes to pre-determine
> the resource identifier of a subsequent call (this can be done via a
> new syscall, or by writing to /proc/self/...).
>
> 2) Each system call that allocates a resource and assigns an identifier
> is modified to check this per-thread field first; if it is set then
> it will attempt to allocate that particular value (if already taken,
> return an error, eg. EBUSY). Otherwise it will proceed as it is today.

But I thought you were just advocating a one-shot filesystem approach for c/r, so we wouldn't be creating the resources piecemeal?

The /proc/self approach is one way to go, it has been working for LSMs this long. I'd agree that it would be nice if we could have a consistent interface to the create_with_id()/set_id() problem. A first shot addressing ipcs and pids would be a great start.

> (I left out some details - eg. the kernel will keep the desire value
> on a per-thread field, when it will be reset, whether we want to also
> tag the field with its type and so on, but the idea is now clear).
>
> The main two advantages are that first, we don't need to devise a new
> method for every syscall that allocates said resources (sigh... just

Agreed.

> think of clone() nightmare to add a new argument);

Yes, and then there will need to be the clone_with_pid() extension on
top of that.

> second, the change
> is incremental: first code the mechanism to set the field, then add
> support in the IPC subsystem, later in the DEVPTS, then in clone and
> so forth.
>
> Oren.
>
> Pierre Peiffer wrote:
>> Kirill Korotaev wrote:
>>> Why user space can need this API? for checkpointing only?
>> I would say "at least for checkpointing"... ;) May be someone else may
>> find an
>> interest about this for something else.
>> In fact, I'm sure that you have some interest in checkpointing; and thus,
>> you
>> have probably some ideas in mind; but whatever the solution you will
>> propose,
>> I'm pretty sure that I could say the same thing for your solution.
>> And what I finally think is: even if it's for "checkpointing only", if
>> many
>> people are interested by this, it may be sufficient to push this ?
>>> Then I would not consider it for inclusion until it is clear how to
>>> implement checkpointing.
>>> As for me personally - I'm against exporting such APIs, since they are
>>> not needed in real-life user space applications and maintaining it
>>> forever for compatibility doesn't worth it.
>> Maintaining these patches is not a big deal, really, but this is not the
>> main
>> point; the "need in real life" (1) is in fact the main one, and then, the
>> "is
>> this solution the best one ?" (2) the second one.
>> About (1), as said in my first mail, as the namespaces and containers are
>> being
>> integrated into the mainline kernel, checkpoint/restart is (or will be)
>> the next
>> need.
>> About (2), my solution propose to do that, as much as possible from
>> userspace,
>> to minimize the kernel impact. Of course, this is subject to discussion.
>> My
>> opinion is that doing a full checkpoint/restart from kernel space will
>> need lot
>> of new specific and intrusive code; I'm not sure that this will be

>> acceptable by
>> the community. But this is my opinion only. Discusion is opened.
>>> Also such APIs allow creation of non-GPL checkpointing in user-space,
>>> which can be of concern as well.
>> Honestly, I don't think this really a concern at all. I mean: I've never
>> seen
>> "this allows non-GPL binary and thus, this is bad" as an argument to
>> reject a
>> functionality, but I may be wrong, and thus, it can be discussed as well.
>> I think the points (1) and (2) as stated above are the key ones.
>> Pierre
>>> Kirill
>>>
>>>
>>> Pierre Peiffer wrote:
>>>> Hi again,
>>>>
>>>> Thinking more about this, I think I must clarify why I choose this way.
>>>> In fact, the idea of these patches is to provide the missing user APIs
>>>> (or
>>>> extend the existing ones) that allow to set or update _all_ properties
>>>> of all
>>>> IPCs, as needed in the case of the checkpoint/restart of an application
>>>> (the
>>>> current user API does not allow to specify an ID for a created IPC, for
>>>> example). And this, without changing the existing API of course.
>>>>
>>>> And msgget(), semget() and shmget() does not have any parameter we can
>>>> use to
>>>> specify an ID.
>>>> That's why I've decided to not change these routines and add a new
>>>> control
>>>> command, IP_SETID, with which we can can change the ID of an IPC. (that
>>>> looks to
>>>> me more straightforward and logical)
>>>>
>>>> Now, this patch is, in fact, only a preparation for the patch 10/15
>>>> which
>>>> really complete the user API by adding this IPC_SETID command.
>>>>
>>>> (... continuing below ...)
>>>>
>>>> Alexey Dobriyan wrote:
>>>>> On Tue, Jan 29, 2008 at 05:02:38PM +0100, pierre.peiffer@bull.net
>>>>> wrote:
>>>>>> This patch provides three new API to change the ID of an existing
>>>>>> System V IPCs.
>>>>>>

>>>>>> These APIs are:
>>>>>> long msg_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>> long sem_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>> long shm_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>>
>>>>>> They return 0 or an error code in case of failure.
>>>>>>
>>>>>> They may be useful for setting a specific ID for an IPC when preparing
>>>>>> a restart operation.
>>>>>>
>>>>>> To be successful, the following rules must be respected:
>>>>>> - the IPC exists (of course...)
>>>>>> - the new ID must satisfy the ID computation rule.
>>>>>> - the entry in the idr corresponding to the new ID must be free.
>>>>>> ipc/util.c | 48
>>>>>> ++
>>>>>> ipc/util.h | 1 +
>>>>>> 8 files changed, 197 insertions(+)
>>>>> For the record, OpenVZ uses "create with predefined ID" method which
>>>>> leads to less code. For example, change at the end is all we want from
>>>>> ipc/util.c .
>>>> And in fact, you do that from kernel space, you don't have the
>>>> constraint to fit
>>>> the existing user API.
>>>> Again, this patch, even if it presents a new kernel API, is in fact a
>>>> preparation for the next patch which introduces a new user API.
>>>>
>>>> Do you think that this could fit your need ?
>>>>
>>>
> ____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID
Posted by Oren Laadan on Wed, 06 Feb 2008 02:06:27 GMT

View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Oren Laadan (orenl@cs.columbia.edu):
>> I strongly second Kirill on this matter.

>>
>> IMHO, we should _avoid_ as much as possible exposing internal kernel
>> state to applications, unless a _real_ need for it is _clearly_
>> demonstrated. The reasons for this are quite obvious.
>
> Hmm, sure, but this sentence is designed to make us want to agree. Yes,
> we want to avoid exporting kernel internals, but generally that means
> things like the precise layout of the task_struct. What Pierre is doing
> is in fact the opposite, exporting resource information in a kernel
> version invariant way.

LOL ... a bit of misunderstanding - let me put some order here:

my response what with respect to the new interface that Pierre
suggested, that is - to add a new IPC call to change an identifier
after it has been allocated (and assigned). This is necessary for the
restart because applications expect to see the same resource id's as
they had at the time of the checkpoint.

What you are referring to is the more recent part of the thread, where
the topic became how data should be saved - in other words, the format
of the checkpoint data. This is entirely orthogonal to my argument.

Now please re-read my email :)

That said, I'd advocate for something in between a raw dump and a pure
"parametric" representation of the data. Raw data tends to be, well,
too raw, which makes the task of reading data from older version by
newer kernels harder to maintain. On the other hand, it is impossible
to abstract everything into kernel-independent format.

>
> In fact, the very reason not to go the route you and Pavel are
> advocating is that if we just dump task state to a file or filesystem
> from the kernel in one shot, we'll be much more tempted to lay out data
> in a way that exports and ends up depending on kernel internals. So
> we'll just want to read and write the task_struct verbatim.
>
> So, there are two very different approaches we can start with.
> Whichever one we follow, we want to avoid having kernel version
> dependencies. They both have their merits to be sure.

You will never be able to avoid that completely, simply because new
kernels will require saving more (or less) data per object, because
of new (or dropped) features.
The best solution in this sense is to provide a filter (hopefully
in user space, utility) that would convert a checkpoint image file
from the old format to a newer format.

And you keep a lot of compatibility code of the kernel, too.

>
> But note that in either case we need to deal with a bunch of locking.
> So getting back to Pierre's patchset, IIRC 1-8 are cleanups worth
> doing no matter 1. 9-11 sound like they are contentuous until
> we decide whether we want to go with a create_with_id() type approach
> or a set_id(). 12 is IMO a good locking cleanup regardless. 13 and
> 15 are contentous until we decide whether we want userspace-controlled
> checkpoint or a one-shot fs. 14 IMO is useful for both c/r approaches.
>
> Is that pretty accurate?

(context switch back to my original reply)

I prefer not to add a new interface to IPC that will provide a new
functionality that isn't needed, except for the checkpoint - because
there is a better alternative to do the same task; this alternative
is more suitable because (a) it can be applied incrementally, (b) it
provides a consistent method to pre-select identifiers of all syscalls,
(where is the current suggestion suggests one way for IPC and will
suggest other hacks for other resources).

(context switch back to the current reply)

I definitely welcome a cleanup of the (insanely multiplexedd) IPC
code. However I argue that the interface need not be extended.

>
>> It isn't strictly necessary to export a new interface in order to
>> support checkpoint/restart. **. Hence, I think that the speculation
>> "we may need it in the future" is too abstract and isn't a good
>> excuse to commit to a new, currently unneeded, interface.
>
> OTOH it did succeed in starting some conversation :)
>
>> Should the
>> need arise in the future, it will be easy to design a new interface
>> (also based on aggregated experience until then).
>
> What aggregated experience? We have to start somewhere...

:) well, assuming the selection of resource IDs is done as I suggested,
we'll have the restart use it. If someone finds a good reason (other
than checkpoint/restart) to pre-select/modify an identifier, it will
be easy to _then_ add an interface. That (hypothetical) interface is
likely to come out more clever after X months using checkpoint/restart.

>
>> ** In fact, the suggested interface may prove problematic (as noted
>> earlier in this thread): if you first create the resource with some
>> arbitrary identifier and then modify the identifier (in our case,
>> IPC id), then the restart procedure is bound to execute sequentially,
>> because of lack of atomicity.
>
> Hmm? Lack of atomicity wrt what? All the tasks being restarted were
> checkpointed at the same time so there will be no conflict in the
> requested IDs, so I don't know what you're referring to.

Consider that we want to have an ultra-fast restart, so we let processes
restart in parallel (as much as possible) in the same container. Task A
wants to allocate IPC id 256, but the kernel allocates 32; before task A
manages to change it to 256 (with the new interface), task B attempts to
create an IPC id 32; the kernel provides, say, 1024, and task B fails to
change it to 32 because it is still used by task A. So restart fails :(

On the other hand, if a process first tells the kernel "I want 32" and
then calls, for instance, semget(), then the IPC can atomically ensure
that the process gets what it wanted.

>
>> That said, I suggest the following method instead (this is the method
>> we use in Zap to determine the desired resource identifier when a new
>> resource is allocated; I recall that we had discussed it in the past,
>> perhaps the mini-summit in september ?):
>>
>> 1) The process/thread tells the kernel that it wishes to pre-determine
>> the resource identifier of a subsequent call (this can be done via a
>> new syscall, or by writing to /proc/self/...).
>>
>> 2) Each system call that allocates a resource and assigns an identifier
>> is modified to check this per-thread field first; if it is set then
>> it will attempt to allocate that particular value (if already taken,
>> return an error, eg. EBUSY). Otherwise it will proceed as it is today.
>
> But I thought you were just advocating a one-shot filesystem approach
> for c/r, so we wouldn't be creating the resources piecemeal?

I wasn't. That was Pavel. While I think the idea is neat, I'm not
convinced that it's practical and best way to go, however I need to
further think about it.

And as I said, I see this as a separate issue from the problem of
create_with_id()/set_id issue().

>

> The /proc/self approach is one way to go, it has been working for LSMs
> this long. I'd agree that it would be nice if we could have a
> consistent interface to the create_with_id()/set_id() problem. A first
> shot addressing ipcs and pids would be a great start.
>
>> (I left out some details - eg. the kernel will keep the desire value
>> on a per-thread field, when it will be reset, whether we want to also
>> tag the field with its type and so on, but the idea is now clear).
>>
>> The main two advantages are that first, we don't need to devise a new
>> method for every syscall that allocates said resources (sigh... just
>
> Agreed.
>
>> think of clone() nightmare to add a new argument);
>
> Yes, and then there will need to be the clone_with_pid() extension on
> top of that.

Exactly ! With the /proc/self/... approach there will not be a need
for a clone_with_pid() extension in terms of user-visible interface;
makes the clone-flags headache a bit more manageable :p

Ah... ok, long one, hopefully clarifies the confusion. That said, I
suggest that the debate regarding the format of the checkpoint data
shall proceed on a new thread, since IMHO it's orthogonal.

Oren.

>
>> second, the change
>> is incremental: first code the mechanism to set the field, then add
>> support in the IPC subsystem, later in the DEVPTS, then in clone and
>> so forth.
>>
>> Oren.
>>
>> Pierre Peiffer wrote:
>>> Kirill Korotaev wrote:
>>>> Why user space can need this API? for checkpointing only?
>>> I would say "at least for checkpointing"... ;) May be someone else may
>>> find an
>>> interest about this for something else.
>>> In fact, I'm sure that you have some interest in checkpointing; and thus,
>>> you
>>> have probably some ideas in mind; but whatever the solution you will
>>> propose,
>>> I'm pretty sure that I could say the same thing for your solution.

>>> And what I finally think is: even if it's for "checkpointing only", if
>>> many
>>> people are interested by this, it may be sufficient to push this ?
>>>> Then I would not consider it for inclusion until it is clear how to
>>>> implement checkpointing.
>>>> As for me personally - I'm against exporting such APIs, since they are
>>>> not needed in real-life user space applications and maintaining it
>>>> forever for compatibility doesn't worth it.
>>> Maintaining these patches is not a big deal, really, but this is not the
>>> main
>>> point; the "need in real life" (1) is in fact the main one, and then, the
>>> "is
>>> this solution the best one ?" (2) the second one.
>>> About (1), as said in my first mail, as the namespaces and containers are
>>> being
>>> integrated into the mainline kernel, checkpoint/restart is (or will be)
>>> the next
>>> need.
>>> About (2), my solution propose to do that, as much as possible from
>>> userspace,
>>> to minimize the kernel impact. Of course, this is subject to discussion.
>>> My
>>> opinion is that doing a full checkpoint/restart from kernel space will
>>> need lot
>>> of new specific and intrusive code; I'm not sure that this will be
>>> acceptable by
>>> the community. But this is my opinion only. Discusion is opened.
>>>> Also such APIs allow creation of non-GPL checkpointing in user-space,
>>>> which can be of concern as well.
>>> Honestly, I don't think this really a concern at all. I mean: I've never
>>> seen
>>> "this allows non-GPL binary and thus, this is bad" as an argument to
>>> reject a
>>> functionality, but I may be wrong, and thus, it can be discussed as well.
>>> I think the points (1) and (2) as stated above are the key ones.
>>> Pierre
>>>> Kirill
>>>>
>>>>
>>>> Pierre Peiffer wrote:
>>>>> Hi again,
>>>>>
>>>>> Thinking more about this, I think I must clarify why I choose this way.
>>>>> In fact, the idea of these patches is to provide the missing user APIs
>>>>> (or
>>>>> extend the existing ones) that allow to set or update _all_ properties
>>>>> of all
>>>>> IPCs, as needed in the case of the checkpoint/restart of an application

>>>>> (the
>>>>> current user API does not allow to specify an ID for a created IPC, for
>>>>> example). And this, without changing the existing API of course.
>>>>>
>>>>> And msgget(), semget() and shmget() does not have any parameter we can
>>>>> use to
>>>>> specify an ID.
>>>>> That's why I've decided to not change these routines and add a new
>>>>> control
>>>>> command, IP_SETID, with which we can can change the ID of an IPC. (that
>>>>> looks to
>>>>> me more straightforward and logical)
>>>>>
>>>>> Now, this patch is, in fact, only a preparation for the patch 10/15
>>>>> which
>>>>> really complete the user API by adding this IPC_SETID command.
>>>>>
>>>>> (... continuing below ...)
>>>>>
>>>>> Alexey Dobriyan wrote:
>>>>>> On Tue, Jan 29, 2008 at 05:02:38PM +0100, pierre.peiffer@bull.net
>>>>>> wrote:
>>>>>>> This patch provides three new API to change the ID of an existing
>>>>>>> System V IPCs.
>>>>>>>
>>>>>>> These APIs are:
>>>>>>> long msg_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>>> long sem_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>>> long shm_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>>>
>>>>>>> They return 0 or an error code in case of failure.
>>>>>>>
>>>>>>> They may be useful for setting a specific ID for an IPC when preparing
>>>>>>> a restart operation.
>>>>>>>
>>>>>>> To be successful, the following rules must be respected:
>>>>>>> - the IPC exists (of course...)
>>>>>>> - the new ID must satisfy the ID computation rule.
>>>>>>> - the entry in the idr corresponding to the new ID must be free.
>>>>>>> ipc/util.c | 48
>>>>>>> ++
>>>>>>> ipc/util.h | 1 +
>>>>>>> 8 files changed, 197 insertions(+)
>>>>>> For the record, OpenVZ uses "create with predefined ID" method which
>>>>>> leads to less code. For example, change at the end is all we want from
>>>>>> ipc/util.c .
>>>>> And in fact, you do that from kernel space, you don't have the
>>>>> constraint to fit

>>>>> the existing user API.
>>>>> Again, this patch, even if it presents a new kernel API, is in fact a
>>>>> preparation for the next patch which introduces a new user API.
>>>>>
>>>>> Do you think that this could fit your need ?
>>>>>
>> ____
>> Containers mailing list
>> Containers@lists.linux-foundation.org
>> https://lists.linux-foundation.org/mailman/listinfo/containers

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID
Posted by serue on Wed, 06 Feb 2008 05:00:39 GMT

View Forum Message <> Reply to Message

Quoting Oren Laadan (orenl@cs.columbia.edu):
>
>
> Serge E. Hallyn wrote:
>> Quoting Oren Laadan (orenl@cs.columbia.edu):
>>> I strongly second Kirill on this matter.
>>>
>>> IMHO, we should _avoid_ as much as possible exposing internal kernel
>>> state to applications, unless a _real_ need for it is _clearly_
>>> demonstrated. The reasons for this are quite obvious.
>> Hmm, sure, but this sentence is designed to make us want to agree. Yes,
>> we want to avoid exporting kernel internals, but generally that means
>> things like the precise layout of the task_struct. What Pierre is doing
>> is in fact the opposite, exporting resource information in a kernel
>> version invariant way.
>
> LOL ... a bit of misunderstanding - let me put some order here:
>
> my response what with respect to the new interface that Pierre
> suggested, that is - to add a new IPC call to change an identifier
> after it has been allocated (and assigned). This is necessary for the
> restart because applications expect to see the same resource id's as
> they had at the time of the checkpoint.
>
> What you are referring to is the more recent part of the thread, where
> the topic became how data should be saved - in other words, the format
> of the checkpoint data. This is entirely orthogonal to my argument.

>
> Now please re-read my email :)

Heh - by the end of my response I was pretty sure that was the case :)

> That said, I'd advocate for something in between a raw dump and a pure
> "parametric" representation of the data. Raw data tends to be, well,
> too raw, which makes the task of reading data from older version by
> newer kernels harder to maintain. On the other hand, it is impossible
> to abstract everything into kernel-independent format.

Well, that's probably getting a little pedantic, but true.

>> In fact, the very reason not to go the route you and Pavel are
>> advocating is that if we just dump task state to a file or filesystem
>> from the kernel in one shot, we'll be much more tempted to lay out data
>> in a way that exports and ends up depending on kernel internals. So
>> we'll just want to read and write the task_struct verbatim.
>> So, there are two very different approaches we can start with.
>> Whichever one we follow, we want to avoid having kernel version
>> dependencies. They both have their merits to be sure.
>
> You will never be able to avoid that completely, simply because new
> kernels will require saving more (or less) data per object, because
> of new (or dropped) features.

Sure.

> The best solution in this sense is to provide a filter (hopefully
> in user space, utility) that would convert a checkpoint image file
> from the old format to a newer format.

Naturally.

> And you keep a lot of compatibility code of the kernel, too.
>
>> But note that in either case we need to deal with a bunch of locking.
>> So getting back to Pierre's patchset, IIRC 1-8 are cleanups worth
>> doing no matter 1. 9-11 sound like they are contentuous until
>> we decide whether we want to go with a create_with_id() type approach
>> or a set_id(). 12 is IMO a good locking cleanup regardless. 13 and
>> 15 are contentous until we decide whether we want userspace-controlled
>> checkpoint or a one-shot fs. 14 IMO is useful for both c/r approaches.
>> Is that pretty accurate?
>
> (context switch back to my original reply)
>
> I prefer not to add a new interface to IPC that will provide a new

> functionality that isn't needed, except for the checkpoint - because
> there is a better alternative to do the same task; this alternative
> is more suitable because (a) it can be applied incrementally, (b) it
> provides a consistent method to pre-select identifiers of all syscalls,
> (where is the current suggestion suggests one way for IPC and will
> suggest other hacks for other resources).
>
> (context switch back to the current reply)
>
> I definitely welcome a cleanup of the (insanely multiplexedd) IPC
> code. However I argue that the interface need not be extended.
>
>>> It isn't strictly necessary to export a new interface in order to
>>> support checkpoint/restart. **. Hence, I think that the speculation
>>> "we may need it in the future" is too abstract and isn't a good
>>> excuse to commit to a new, currently unneeded, interface.
>> OTOH it did succeed in starting some conversation :)
>>> Should the
>>> need arise in the future, it will be easy to design a new interface
>>> (also based on aggregated experience until then).
>> What aggregated experience? We have to start somewhere...
>
> :) well, assuming the selection of resource IDs is done as I suggested,
> we'll have the restart use it. If someone finds a good reason (other
> than checkpoint/restart) to pre-select/modify an identifier, it will
> be easy to _then_ add an interface. That (hypothetical) interface is
> likely to come out more clever after X months using checkpoint/restart.
>
>>> ** In fact, the suggested interface may prove problematic (as noted
>>> earlier in this thread): if you first create the resource with some
>>> arbitrary identifier and then modify the identifier (in our case,
>>> IPC id), then the restart procedure is bound to execute sequentially,
>>> because of lack of atomicity.
>> Hmm? Lack of atomicity wrt what? All the tasks being restarted were
>> checkpointed at the same time so there will be no conflict in the
>> requested IDs, so I don't know what you're referring to.
>
> Consider that we want to have an ultra-fast restart, so we let processes
> restart in parallel (as much as possible) in the same container. Task A
> wants to allocate IPC id 256, but the kernel allocates 32; before task A
> manages to change it to 256 (with the new interface), task B attempts to
> create an IPC id 32; the kernel provides, say, 1024, and task B fails to
> change it to 32 because it is still used by task A. So restart fails :(

Bah, it gets -EAGAIN and tries again. I see the biggest plus of your
approach as being the consistent api.

> On the other hand, if a process first tells the kernel "I want 32" and

> then calls, for instance, semget(), then the IPC can atomically ensure
> that the process gets what it wanted.
>
>>> That said, I suggest the following method instead (this is the method
>>> we use in Zap to determine the desired resource identifier when a new
>>> resource is allocated; I recall that we had discussed it in the past,
>>> perhaps the mini-summit in september ?):
>>>
>>> 1) The process/thread tells the kernel that it wishes to pre-determine
>>> the resource identifier of a subsequent call (this can be done via a
>>> new syscall, or by writing to /proc/self/...).
>>>
>>> 2) Each system call that allocates a resource and assigns an identifier
>>> is modified to check this per-thread field first; if it is set then
>>> it will attempt to allocate that particular value (if already taken,
>>> return an error, eg. EBUSY). Otherwise it will proceed as it is today.
>> But I thought you were just advocating a one-shot filesystem approach
>> for c/r, so we wouldn't be creating the resources piecemeal?
>
> I wasn't. That was Pavel. While I think the idea is neat, I'm not
> convinced that it's practical and best way to go, however I need to
> further think about it.
>
> And as I said, I see this as a separate issue from the problem of
> create_with_id()/set_id issue().
>
>> The /proc/self approach is one way to go, it has been working for LSMs
>> this long. I'd agree that it would be nice if we could have a
>> consistent interface to the create_with_id()/set_id() problem. A first
>> shot addressing ipcs and pids would be a great start.
>>> (I left out some details - eg. the kernel will keep the desire value
>>> on a per-thread field, when it will be reset, whether we want to also
>>> tag the field with its type and so on, but the idea is now clear).
>>>
>>> The main two advantages are that first, we don't need to devise a new
>>> method for every syscall that allocates said resources (sigh... just
>> Agreed.
>>> think of clone() nightmare to add a new argument);
>> Yes, and then there will need to be the clone_with_pid() extension on
>> top of that.
>
> Exactly ! With the /proc/self/... approach there will not be a need
> for a clone_with_pid() extension in terms of user-visible interface;
> makes the clone-flags headache a bit more manageable :p

So you say this is how zap does it now? Would it be pretty trivial to
make a small patch consisting of your base procpid code and the clone
plugin to let you clone with a particular pid, and post that?

thanks,
-serge

> Ah... ok, long one, hopefully clarifies the confusion. That said, I
> suggest that the debate regarding the format of the checkpoint data
> shall proceed on a new thread, since IMHO it's orthogonal.
>
> Oren.
>
>>> second, the change
>>> is incremental: first code the mechanism to set the field, then add
>>> support in the IPC subsystem, later in the DEVPTS, then in clone and
>>> so forth.
>>>
>>> Oren.
>>>
>>> Pierre Peiffer wrote:
>>>> Kirill Korotaev wrote:
>>>>> Why user space can need this API? for checkpointing only?
>>>> I would say "at least for checkpointing"... ;) May be someone else may
>>>> find an
>>>> interest about this for something else.
>>>> In fact, I'm sure that you have some interest in checkpointing; and
>>>> thus, you
>>>> have probably some ideas in mind; but whatever the solution you will
>>>> propose,
>>>> I'm pretty sure that I could say the same thing for your solution.
>>>> And what I finally think is: even if it's for "checkpointing only", if
>>>> many
>>>> people are interested by this, it may be sufficient to push this ?
>>>>> Then I would not consider it for inclusion until it is clear how to
>>>>> implement checkpointing.
>>>>> As for me personally - I'm against exporting such APIs, since they are
>>>>> not needed in real-life user space applications and maintaining it
>>>>> forever for compatibility doesn't worth it.
>>>> Maintaining these patches is not a big deal, really, but this is not the
>>>> main
>>>> point; the "need in real life" (1) is in fact the main one, and then,
>>>> the "is
>>>> this solution the best one ?" (2) the second one.
>>>> About (1), as said in my first mail, as the namespaces and containers
>>>> are being
>>>> integrated into the mainline kernel, checkpoint/restart is (or will be)
>>>> the next
>>>> need.
>>>> About (2), my solution propose to do that, as much as possible from
>>>> userspace,

>>>> to minimize the kernel impact. Of course, this is subject to discussion.
>>>> My
>>>> opinion is that doing a full checkpoint/restart from kernel space will
>>>> need lot
>>>> of new specific and intrusive code; I'm not sure that this will be
>>>> acceptable by
>>>> the community. But this is my opinion only. Discusion is opened.
>>>>> Also such APIs allow creation of non-GPL checkpointing in user-space,
>>>>> which can be of concern as well.
>>>> Honestly, I don't think this really a concern at all. I mean: I've never
>>>> seen
>>>> "this allows non-GPL binary and thus, this is bad" as an argument to
>>>> reject a
>>>> functionality, but I may be wrong, and thus, it can be discussed as
>>>> well.
>>>> I think the points (1) and (2) as stated above are the key ones.
>>>> Pierre
>>>>> Kirill
>>>>>
>>>>>
>>>>> Pierre Peiffer wrote:
>>>>>> Hi again,
>>>>>>
>>>>>> Thinking more about this, I think I must clarify why I choose this
>>>>>> way.
>>>>>> In fact, the idea of these patches is to provide the missing user APIs
>>>>>> (or
>>>>>> extend the existing ones) that allow to set or update _all_ properties
>>>>>> of all
>>>>>> IPCs, as needed in the case of the checkpoint/restart of an
>>>>>> application (the
>>>>>> current user API does not allow to specify an ID for a created IPC,
>>>>>> for
>>>>>> example). And this, without changing the existing API of course.
>>>>>>
>>>>>> And msgget(), semget() and shmget() does not have any parameter we
>>>>>> can use to
>>>>>> specify an ID.
>>>>>> That's why I've decided to not change these routines and add a new
>>>>>> control
>>>>>> command, IP_SETID, with which we can can change the ID of an IPC.
>>>>>> (that looks to
>>>>>> me more straightforward and logical)
>>>>>>
>>>>>> Now, this patch is, in fact, only a preparation for the patch 10/15
>>>>>> which
>>>>>> really complete the user API by adding this IPC_SETID command.
>>>>>>

>>>>>> (... continuing below ...)
>>>>>>
>>>>>> Alexey Dobriyan wrote:
>>>>>>> On Tue, Jan 29, 2008 at 05:02:38PM +0100, pierre.peiffer@bull.net
>>>>>>> wrote:
>>>>>>>> This patch provides three new API to change the ID of an existing
>>>>>>>> System V IPCs.
>>>>>>>>
>>>>>>>> These APIs are:
>>>>>>>> long msg_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>>>> long sem_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>>>> long shm_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>>>>
>>>>>>>> They return 0 or an error code in case of failure.
>>>>>>>>
>>>>>>>> They may be useful for setting a specific ID for an IPC when
>>>>>>>> preparing
>>>>>>>> a restart operation.
>>>>>>>>
>>>>>>>> To be successful, the following rules must be respected:
>>>>>>>> - the IPC exists (of course...)
>>>>>>>> - the new ID must satisfy the ID computation rule.
>>>>>>>> - the entry in the idr corresponding to the new ID must be free.
>>>>>>>> ipc/util.c | 48
>>>>>>>> ++
>>>>>>>> ipc/util.h | 1 +
>>>>>>>> 8 files changed, 197 insertions(+)
>>>>>>> For the record, OpenVZ uses "create with predefined ID" method which
>>>>>>> leads to less code. For example, change at the end is all we want
>>>>>>> from
>>>>>>> ipc/util.c .
>>>>>> And in fact, you do that from kernel space, you don't have the
>>>>>> constraint to fit
>>>>>> the existing user API.
>>>>>> Again, this patch, even if it presents a new kernel API, is in fact a
>>>>>> preparation for the next patch which introduces a new user API.
>>>>>>
>>>>>> Do you think that this could fit your need ?
>>>>>>
>>> ____
>>> Containers mailing list
>>> Containers@lists.linux-foundation.org
>>> https://lists.linux-foundation.org/mailman/listinfo/containers
____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID
Posted by Pierre Peiffer on Fri, 08 Feb 2008 10:12:33 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
>
> But note that in either case we need to deal with a bunch of locking.
> So getting back to Pierre's patchset, IIRC 1-8 are cleanups worth
> doing no matter 1. 9-11 sound like they are contentuous until
> we decide whether we want to go with a create_with_id() type approach
> or a set_id(). 12 is IMO a good locking cleanup regardless. 13 and
> 15 are contentous until we decide whether we want userspace-controlled
> checkpoint or a one-shot fs. 14 IMO is useful for both c/r approaches.
>
> Is that pretty accurate?
>

Ok, so, so far, the discussion stays opened about the new functionalities for c/r.

As there were no objection about the first patches, which rewrite/enhance the
existing code, Andrew, could you consider them (ie patches 1 to 8 of this
series) for inclusion in -mm ? (I mean, as soon as it is possible, as I guess
you're pretty busy for now with the merge for 2.6.25)

If you prefer, I can resend them separately ?

Thanks,

Pierre

>> It isn't strictly necessary to export a new interface in order to
>> support checkpoint/restart. **. Hence, I think that the speculation
>> "we may need it in the future" is too abstract and isn't a good
>> excuse to commit to a new, currently unneeded, interface.
>
> OTOH it did succeed in starting some conversation :)
>
>> Should the
>> need arise in the future, it will be easy to design a new interface
>> (also based on aggregated experience until then).
>
> What aggregated experience? We have to start somewhere...
>
>> ** In fact, the suggested interface may prove problematic (as noted
>> earlier in this thread): if you first create the resource with some
>> arbitrary identifier and then modify the identifier (in our case,
>> IPC id), then the restart procedure is bound to execute sequentially,

>> because of lack of atomicity.
>
> Hmm? Lack of atomicity wrt what? All the tasks being restarted were
> checkpointed at the same time so there will be no conflict in the
> requested IDs, so I don't know what you're referring to.
>
>> That said, I suggest the following method instead (this is the method
>> we use in Zap to determine the desired resource identifier when a new
>> resource is allocated; I recall that we had discussed it in the past,
>> perhaps the mini-summit in september ?):
>>
>> 1) The process/thread tells the kernel that it wishes to pre-determine
>> the resource identifier of a subsequent call (this can be done via a
>> new syscall, or by writing to /proc/self/...).
>>
>> 2) Each system call that allocates a resource and assigns an identifier
>> is modified to check this per-thread field first; if it is set then
>> it will attempt to allocate that particular value (if already taken,
>> return an error, eg. EBUSY). Otherwise it will proceed as it is today.
>
> But I thought you were just advocating a one-shot filesystem approach
> for c/r, so we wouldn't be creating the resources piecemeal?
>
> The /proc/self approach is one way to go, it has been working for LSMs
> this long. I'd agree that it would be nice if we could have a
> consistent interface to the create_with_id()/set_id() problem. A first
> shot addressing ipcs and pids would be a great start.
>
>> (I left out some details - eg. the kernel will keep the desire value
>> on a per-thread field, when it will be reset, whether we want to also
>> tag the field with its type and so on, but the idea is now clear).
>>
>> The main two advantages are that first, we don't need to devise a new
>> method for every syscall that allocates said resources (sigh... just
>
> Agreed.
>
>> think of clone() nightmare to add a new argument);
>
> Yes, and then there will need to be the clone_with_pid() extension on
> top of that.
>
>> second, the change
>> is incremental: first code the mechanism to set the field, then add
>> support in the IPC subsystem, later in the DEVPTS, then in clone and
>> so forth.
>>
>> Oren.

>>
>> Pierre Peiffer wrote:
>>> Kirill Korotaev wrote:
>>>> Why user space can need this API? for checkpointing only?
>>> I would say "at least for checkpointing"... ;) May be someone else may
>>> find an
>>> interest about this for something else.
>>> In fact, I'm sure that you have some interest in checkpointing; and thus,
>>> you
>>> have probably some ideas in mind; but whatever the solution you will
>>> propose,
>>> I'm pretty sure that I could say the same thing for your solution.
>>> And what I finally think is: even if it's for "checkpointing only", if
>>> many
>>> people are interested by this, it may be sufficient to push this ?
>>>> Then I would not consider it for inclusion until it is clear how to
>>>> implement checkpointing.
>>>> As for me personally - I'm against exporting such APIs, since they are
>>>> not needed in real-life user space applications and maintaining it
>>>> forever for compatibility doesn't worth it.
>>> Maintaining these patches is not a big deal, really, but this is not the
>>> main
>>> point; the "need in real life" (1) is in fact the main one, and then, the
>>> "is
>>> this solution the best one ?" (2) the second one.
>>> About (1), as said in my first mail, as the namespaces and containers are
>>> being
>>> integrated into the mainline kernel, checkpoint/restart is (or will be)
>>> the next
>>> need.
>>> About (2), my solution propose to do that, as much as possible from
>>> userspace,
>>> to minimize the kernel impact. Of course, this is subject to discussion.
>>> My
>>> opinion is that doing a full checkpoint/restart from kernel space will
>>> need lot
>>> of new specific and intrusive code; I'm not sure that this will be
>>> acceptable by
>>> the community. But this is my opinion only. Discusion is opened.
>>>> Also such APIs allow creation of non-GPL checkpointing in user-space,
>>>> which can be of concern as well.
>>> Honestly, I don't think this really a concern at all. I mean: I've never
>>> seen
>>> "this allows non-GPL binary and thus, this is bad" as an argument to
>>> reject a
>>> functionality, but I may be wrong, and thus, it can be discussed as well.
>>> I think the points (1) and (2) as stated above are the key ones.
>>> Pierre

>>>> Kirill
>>>>
>>>>
>>>> Pierre Peiffer wrote:
>>>>> Hi again,
>>>>>
>>>>> Thinking more about this, I think I must clarify why I choose this way.
>>>>> In fact, the idea of these patches is to provide the missing user APIs
>>>>> (or
>>>>> extend the existing ones) that allow to set or update _all_ properties
>>>>> of all
>>>>> IPCs, as needed in the case of the checkpoint/restart of an application
>>>>> (the
>>>>> current user API does not allow to specify an ID for a created IPC, for
>>>>> example). And this, without changing the existing API of course.
>>>>>
>>>>> And msgget(), semget() and shmget() does not have any parameter we can
>>>>> use to
>>>>> specify an ID.
>>>>> That's why I've decided to not change these routines and add a new
>>>>> control
>>>>> command, IP_SETID, with which we can can change the ID of an IPC. (that
>>>>> looks to
>>>>> me more straightforward and logical)
>>>>>
>>>>> Now, this patch is, in fact, only a preparation for the patch 10/15
>>>>> which
>>>>> really complete the user API by adding this IPC_SETID command.
>>>>>
>>>>> (... continuing below ...)
>>>>>
>>>>> Alexey Dobriyan wrote:
>>>>>> On Tue, Jan 29, 2008 at 05:02:38PM +0100, pierre.peiffer@bull.net
>>>>>> wrote:
>>>>>>> This patch provides three new API to change the ID of an existing
>>>>>>> System V IPCs.
>>>>>>>
>>>>>>> These APIs are:
>>>>>>> long msg_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>>> long sem_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>>> long shm_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>>>
>>>>>>> They return 0 or an error code in case of failure.
>>>>>>>
>>>>>>> They may be useful for setting a specific ID for an IPC when preparing
>>>>>>> a restart operation.
>>>>>>>
>>>>>>> To be successful, the following rules must be respected:

>>>>>>> - the IPC exists (of course...)
>>>>>>> - the new ID must satisfy the ID computation rule.
>>>>>>> - the entry in the idr corresponding to the new ID must be free.
>>>>>>> ipc/util.c | 48
>>>>>>> ++
>>>>>>> ipc/util.h | 1 +
>>>>>>> 8 files changed, 197 insertions(+)
>>>>>> For the record, OpenVZ uses "create with predefined ID" method which
>>>>>> leads to less code. For example, change at the end is all we want from
>>>>>> ipc/util.c .
>>>>> And in fact, you do that from kernel space, you don't have the
>>>>> constraint to fit
>>>>> the existing user API.
>>>>> Again, this patch, even if it presents a new kernel API, is in fact a
>>>>> preparation for the next patch which introduces a new user API.
>>>>>
>>>>> Do you think that this could fit your need ?
>>>>>
>> ____
>> Containers mailing list
>> Containers@lists.linux-foundation.org
>> https://lists.linux-foundation.org/mailman/listinfo/containers
>
>

--
Pierre Peiffer

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers