Subject: [RFC PATCH 3/4] IPC/sem: prepare semundo code to work on a third party task
Posted by Nadia Derbey on Fri, 20 Jun 2008 11:48:41 GMT
View Forum Message <> Reply to Message

PATCH [03/04]

In order to change a task's semundo-list from procfs, we must be able to
work on any target task.
But the existing code that manages the semundo-list, currently only works
on the 'current' task, not allowing to specify a target task.

This patch changes all these routines to make them 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>
Signed-off-by: Nadia Derbey <Nadia.Derbey@bull.net>

---
 ipc/sem.c | 116 ++----------
 1 file changed, 83 insertions(+), 33 deletions(-)

Index: linux-2.6.26-rc5-mm3/ipc/sem.c
===
--- linux-2.6.26-rc5-mm3.orig/ipc/sem.c 2008-06-20 12:01:55.000000000 +0200
+++ linux-2.6.26-rc5-mm3/ipc/sem.c 2008-06-20 12:43:33.000000000 +0200
@@ -925,8 +925,9 @@ 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.
@@ -934,28 +935,68 @@ asmlinkage long sys_semctl (int semid, i
  * at exit time.
  *
  * This can block, so callers must hold no locks.
+ *
+ * Note:
+ * If there is already an undo_list for this task, there is no need

```
+ * to hold the task-lock to retrieve it, as the pointer can not change
+ * afterwards.
+ *
+ * Concurrent tasks are allowed to access and go through the semundo_list
+ * only if they successfully grabbed a refcnt.
+ * If the undo_list is created here, its refcnt is unconditionally set to 2.
  */
-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;

+ rcu_read_lock();
+ undo_list = rcu_dereference(tsk->sysvsem.undo_list);
  /*
-  * No need to have a rcu read critical section here: noone but current
-  * is accessing the undo_list.
+  * In order for the rcu protection in exit_sem() to work, increment
+  * the refcount on the undo_list within the same rcu cycle in
+  * which it rcu_dereferenced() the undo_list from the task_struct.
   */
- undo_list = current->sysvsem.undo_list;
+ if (undo_list)
+  atomic_inc(&undo_list->refcnt);
+ rcu_read_unlock();
  if (!undo_list) {
   undo_list = kzalloc(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) {
+   kfree(undo_list);
+   undo_list = tsk->sysvsem.undo_list;
+   task_unlock(tsk);
+   goto out;
+  }
+
   spin_lock_init(&undo_list->lock);
-  atomic_set(&undo_list->refcnt, 1);
-  undo_list->ns = get_ipc_ns(current->nsproxy->ipc_ns);
+
+  /*
+   * get_undo_list can be called from the following routines:
+   * 1) copy_semundo:
```

```
+   *     the refcnt must be set to 2 (1 for the parent and 1 for
+   *     the child.
+   * 2) sys_semtimedop:
+   *     will decrement the refcnt after calling get_undo_list
+   *     so set it to 2 in order for the undo_list to be kept
+   */
+  atomic_set(&undo_list->refcnt, 2);
+
+  undo_list->ns = get_ipc_ns(tsk->nsproxy->ipc_ns);
    INIT_LIST_HEAD(&undo_list->list_proc);

-  rcu_assign_pointer(current->sysvsem.undo_list, undo_list);
+  rcu_assign_pointer(tsk->sysvsem.undo_list, undo_list);
+
+  task_unlock(tsk);
   }
- *undo_listp = undo_list;
+out:
+ *ulp = undo_list;
  return 0;
 }

@@ -972,7 +1013,7 @@ static struct sem_undo *lookup_undo(stru

 /**
  * find_alloc_undo - Lookup (and if not present create) undo array
- * @ns: namespace
+ * @ulp: undo list pointer (already created if it was not present)
  * @semid: semaphore array id
  *
  * The function looks up (and if not present creates) the undo structure.
@@ -981,17 +1022,12 @@ static struct sem_undo *lookup_undo(stru
  * Lifetime-rules: sem_undo is rcu-protected, on success, the function
  * performs a rcu_read_lock().
  */
-static struct sem_undo *find_alloc_undo(struct ipc_namespace *ns, int semid)
+static struct sem_undo *find_alloc_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);
```

```
  rcu_read_lock();
  spin_lock(&ulp->lock);
@@ -1001,6 +1037,8 @@ static struct sem_undo *find_alloc_undo(
   goto out;
  rcu_read_unlock();

+ ns = ulp->ns;
+
  /* no undo structure around - allocate one. */
  /* step 1: figure out the size of the semaphore array */
  sma = sem_lock_check(ns, semid);
@@ -1060,6 +1098,7 @@ 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;
@@ -1104,11 +1143,15 @@ asmlinkage long sys_semtimedop(int semid
    alter = 1;
  }

+ error = get_undo_list(current, &ulp);
+ if (error)
+  goto out_free;
+
  if (undos) {
-  un = find_alloc_undo(ns, semid);
+  un = find_alloc_undo(ulp, semid);
   if (IS_ERR(un)) {
    error = PTR_ERR(un);
-   goto out_free;
+   goto out_put_ulp;
   }
  } else
   un = NULL;
@@ -1118,11 +1161,11 @@ asmlinkage long sys_semtimedop(int semid
   if (un)
    rcu_read_unlock();
   error = PTR_ERR(sma);
-  goto out_free;
+  goto out_put_ulp;
  }

  /*
-  * semid identifiers are not unique - find_alloc_undo may have
```

```
+  * semid identifiers are not unique - get_undo_list may have
   * allocated an undo structure, it was invalidated by an RMID
   * and now a new array with received the same id. Check and fail.
   * This case can be detected checking un->semid. The existance of
@@ -1225,6 +1268,9 @@ asmlinkage long sys_semtimedop(int semid

 out_unlock_free:
  sem_unlock(sma);
+out_put_ulp:
+ atomic_dec_and_test(&ulp->refcnt);
+ BUG_ON(!atomic_read(&ulp->refcnt));
 out_free:
  if(sops != fast_sops)
   kfree(sops);
@@ -1246,10 +1292,9 @@ 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);
   tsk->sysvsem.undo_list = undo_list;
  } else
   tsk->sysvsem.undo_list = NULL;
@@ -1258,7 +1303,8 @@ 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
@@ -1271,6 +1317,8 @@ int copy_semundo(unsigned long clone_fla
 static void free_semundo_list(struct sem_undo_list *ulp)
 {
+ BUG_ON(atomic_read(&ulp->refcnt));
+
  for (;;) {
   struct sem_array *sma;
   struct sem_undo *un;
@@ -1346,26 +1394,28 @@ static void free_semundo_list(struct sem
  }
  put_ipc_ns(ulp->ns);
```

```
  /*
-  * No need to call synchronize_rcu() here: we come here if the refcnt
-  * is 0 and this has been done into exit_sem after synchronizing. So
-  * nobody else can be referencing to the undo_list.
+  * We are here if the refcnt became 0, so only a single task can be
+  * accessing that undo_list.
   */
  kfree(ulp);
 }

-/* called from do_exit() */
+/* called from do_exit()
+ * task_lock() is used to synchronize between the undo_list creation
+ * (in get_undo_list()) and its removal.
+ */
 void exit_sem(struct task_struct *tsk)
 {
  struct sem_undo_list *ulp;

- rcu_read_lock();
- ulp = rcu_dereference(tsk->sysvsem.undo_list);
+ task_lock(tsk);
+ ulp = tsk->sysvsem.undo_list;
  if (!ulp) {
-  rcu_read_unlock();
+  task_unlock(tsk);
   return;
  }
- rcu_read_unlock();
  rcu_assign_pointer(tsk->sysvsem.undo_list, NULL);
+ task_unlock(tsk);
  synchronize_rcu();

  if (atomic_dec_and_test(&ulp->refcnt))

--
```

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