
Subject: [RFC PATCH 1/4] IPC/sem: use RCU to free the sem_undo_list
Posted by [Nadia Derby](#) on Fri, 20 Jun 2008 11:48:39 GMT
[View Forum Message](#) <> [Reply to Message](#)

PATCH [01/04]

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>

Signed-off-by: Nadia Derby <Nadia.Derbey@bull.net>

include/linux/sem.h | 5 ++++-
ipc/sem.c | 46 ++++++-----
2 files changed, 37 insertions(+), 14 deletions(-)

Index: linux-2.6.26-rc5-mm3/include/linux/sem.h

```
=====
--- linux-2.6.26-rc5-mm3.orig/include/linux/sem.h 2008-06-20 11:11:39.000000000 +0200
+++ linux-2.6.26-rc5-mm3/include/linux/sem.h 2008-06-20 11:11:21.000000000 +0200
@@ -112,7 +112,8 @@ 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 list_head list_proc; /* per-process list: all undos from one process. */
@@ -131,6 +132,8 @@ struct sem_undo_list {
    atomic_t refcnt;
    spinlock_t lock;
    struct list_head list_proc;
+ struct ipc_namespace *ns;
+ struct rcu_head rcu;
};
```

```
struct sysv_sem {
```

Index: linux-2.6.26-rc5-mm3/ipc/sem.c

=====

--- linux-2.6.26-rc5-mm3.orig/ipc/sem.c 2008-06-20 11:11:56.000000000 +0200

+++ linux-2.6.26-rc5-mm3/ipc/sem.c 2008-06-20 12:01:45.000000000 +0200

@@ -939,6 +939,10 @@ static inline int get_undo_list(struct s

```
{
    struct sem_undo_list *undo_list;
```

```
+ /*
+ * No need to have a rcu read critical section here: noone but current
+ * is accessing the undo_list.
+ */
```

```
    undo_list = current->sysvsem.undo_list;
    if (!undo_list) {
        undo_list = kzalloc(sizeof(*undo_list), GFP_KERNEL);
@@ -946,9 +950,10 @@ 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);
    INIT_LIST_HEAD(&undo_list->list_proc);
```

```
- current->sysvsem.undo_list = undo_list;
+ rcu_assign_pointer(current->sysvsem.undo_list, undo_list);
}
*undo_listp = undo_list;
return 0;
```

@@ -1264,18 +1269,8 @@ int copy_semundo(unsigned long clone_flg

```
* The current implementation does not do so. The POSIX standard
* and SVID should be consulted to determine what behavior is mandated.
```

```

*/
-void exit_sem(struct task_struct *tsk)
+static void free_semundo_list(struct sem_undo_list *ulp)
{
- struct sem_undo_list *ulp;
-
- ulp = tsk->sysvsem.undo_list;
- if (!ulp)
- return;
- tsk->sysvsem.undo_list = NULL;
-
- if (!atomic_dec_and_test(&ulp->refcnt))
- return;
-
  for (;;) {
    struct sem_array *sma;
    struct sem_undo *un;
@@ -1294,7 +1289,7 @@ void exit_sem(struct task_struct *tsk)
    if (semid == -1)
      break;

- sma = sem_lock_check(tsk->nsproxy->ipc_ns, un->semid);
+ sma = sem_lock_check(ulp->ns, un->semid);

    /* exit_sem raced with IPC_RMID, nothing to do */
    if (IS_ERR(sma))
@@ -1349,9 +1344,34 @@ void exit_sem(struct task_struct *tsk)

    call_rcu(&un->rcu, free_un);
  }
+ 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.
+ */
  kfree(ulp);
}

+/* called from do_exit() */
+void exit_sem(struct task_struct *tsk)
+{
+ struct sem_undo_list *ulp;
+
+ rcu_read_lock();
+ ulp = rcu_dereference(tsk->sysvsem.undo_list);
+ if (!ulp) {
+ rcu_read_unlock();

```

```
+ return;
+ }
+ rcu_read_unlock();
+ rcu_assign_pointer(tsk->sysvsem.undo_list, NULL);
+ synchronize_rcu();
+
+ if (atomic_dec_and_test(&ulp->refcnt))
+   free_semundo_list(ulp);
+}
+
#ifdef CONFIG_PROC_FS
static int sysvipc_sem_proc_show(struct seq_file *s, void *it)
{

--
```

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