Subject: [RFC PATCH 0/4] IPC/sem - allow saving/restoring a process semundo_list
Posted by Nadia Derbey on Fri, 20 Jun 2008 11:48:38 GMT
View Forum Message <> Reply to Message

Hi,

A couple of months ago, Pierre Peiffer has submitted a patch series to enable checkpointing / restarting ipcs.
4 of these patches were related to semaphores:

https://lists.linux-foundation.org/pipermail/containers/2008-January/thread.html#9756
https://lists.linux-foundation.org/pipermail/containers/2008-January/thread.html#9757
https://lists.linux-foundation.org/pipermail/containers/2008-January/thread.html#9758
https://lists.linux-foundation.org/pipermail/containers/2008-January/thread.html#9759

They introduced a new procfs file: /proc/<pid>/semundo to read and write the semaphores undo values for a given process.

These patches are widely used in the -lxc development tree cryo code is based upon.
(more information about lxc can be found at http://lxc.sourceforge.net/ and the development tree can be found there too - under the patches link).

Manfred Spraul, on his side, has rewritten an important part of the semaphores code. See:

1. http://lkml.org/lkml/2008/5/24/92
2. http://lkml.org/lkml/2008/5/24/93
3. http://lkml.org/lkml/2008/5/24/90
4. http://lkml.org/lkml/2008/5/24/91

Mainly patches 1 and 4 made Pierre's patches unappliable.
 . patch 1 changed the semundo_list proc_list field into a linked list.
 . patch 2
    . reversed the locking order of the sem_undo_list lock and the semaphore lock.
    . converted the sem_undo structure to use rcu.

Since 2.6.26-rc5-mm3 is now the new target for the -lxc development tree, I've ported them and taking this opportunity to resubmit them to the containers list.

I have in mind a simpler solution, which I think I'll propose next week:
instead of writing into the /proc/<pid>/semundo of a third party process, only allow the write operation to be done for <current>.

Kathy, can you please pull in these patches?

Regards,
Nadia

--

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

---

Subject: [RFC PATCH 1/4] IPC/sem: use RCU to free the sem_undo_list
Posted by Nadia Derbey 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 Derbey <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_fla
  * 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

---

## Subject: [RFC PATCH 2/4] IPC/sem: per <pid> semundo file in procfs
Posted by Nadia Derbey on Fri, 20 Jun 2008 11:48:40 GMT
View Forum Message <> Reply to Message

PATCH [02/04]

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 particularly useful to allow a user access
these data, for example for checkpointing a process

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

---
 fs/proc/base.c     |    3
 fs/proc/internal.h |    1
 ipc/sem.c          |  163 ++++++++++++++++++++++++++++++++++++++++++++++++++++
 3 files changed, 167 insertions(+)

Index: linux-2.6.26-rc5-mm3/fs/proc/base.c
===================================================================
--- linux-2.6.26-rc5-mm3.orig/fs/proc/base.c 2008-06-20 12:01:19.000000000 +0200
+++ linux-2.6.26-rc5-mm3/fs/proc/base.c 2008-06-20 12:01:55.000000000 +0200
@@ -2525,6 +2525,9 @@ static const struct pid_entry tgid_base_
 #ifdef CONFIG_TASK_IO_ACCOUNTING
  INF("io", S_IRUGO, tgid_io_accounting),
 #endif
+#ifdef CONFIG_SYSVIPC
+ REG("semundo",    S_IRUGO, semundo),
+#endif
 };

 static int proc_tgid_base_readdir(struct file * filp,
Index: linux-2.6.26-rc5-mm3/fs/proc/internal.h
===================================================================
--- linux-2.6.26-rc5-mm3.orig/fs/proc/internal.h 2008-06-20 12:01:19.000000000 +0200
+++ linux-2.6.26-rc5-mm3/fs/proc/internal.h 2008-06-20 12:01:55.000000000 +0200
@@ -65,6 +65,7 @@ extern const struct file_operations proc
 extern const struct file_operations proc_net_operations;
 extern const struct file_operations proc_kmsg_operations;
 extern const struct inode_operations proc_net_inode_operations;
+extern const struct file_operations proc_semundo_operations;

 void free_proc_entry(struct proc_dir_entry *de);

Index: linux-2.6.26-rc5-mm3/ipc/sem.c
===================================================================
--- linux-2.6.26-rc5-mm3.orig/ipc/sem.c 2008-06-20 12:01:45.000000000 +0200
+++ linux-2.6.26-rc5-mm3/ipc/sem.c 2008-06-20 12:01:55.000000000 +0200
@@ -1390,4 +1390,167 @@ static int sysvipc_sem_proc_show(struct
     sma->sem_otime,
     sma->sem_ctime);
 }
+
+/* iterator */

```
+/* The rcu_read_lock is kept from the .start to the .stop routines */
+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 from being
+  * freed.
+  */
+ rcu_read_lock();
+ spin_lock(&undo_list->lock);
+ list_for_each_entry_rcu(undo, &undo_list->list_proc, list_proc) {
+  if ((undo->semid != -1) && !(pos--))
+   break;
+ }
+ spin_unlock(&undo_list->lock);
+
+ if (&undo->list_proc == &undo_list->list_proc)
+  return NULL;
+
+ 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 = list_entry(rcu_dereference(undo_list->list_proc.next),
```

```
+    struct sem_undo, list_proc);
+
+ } while (&undo->list_proc != &undo_list->list_proc
+       && undo->semid == -1);
+
+ ++*ppos;
+ spin_unlock(&undo_list->lock);
+
+ if (&undo->list_proc == &undo_list->list_proc)
+  return NULL;
+ return undo;
+}
+
+static void semundo_stop(struct seq_file *m, void *v)
+{
+ rcu_read_unlock();
+}
+
+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

--
```

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

---

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

---

Subject: [RFC PATCH 4/4] IPC/sem: add the write() operation to the semundo file in procfs
Posted by Nadia Derbey on Fri, 20 Jun 2008 11:48:42 GMT
View Forum Message <> Reply to Message

PATCH [04/04]

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

The user must provide a line per semaphore. Each line containes the
semaphore ID followed by the semaphores values to undo.

The operation failes 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>
Signed-off-by: Nadia Derbey <Nadia.Derbey@bull.net>

---
```
 fs/proc/base.c      |   2
 include/linux/sem.h |   3
 ipc/sem.c           | 278 ++++++++++++++++++++++++++++++++++++++++++++++++++--
 3 files changed, 274 insertions(+), 9 deletions(-)

Index: linux-2.6.26-rc5-mm3/fs/proc/base.c
===================================================================
```

```
--- linux-2.6.26-rc5-mm3.orig/fs/proc/base.c 2008-06-20 12:01:55.000000000 +0200
+++ linux-2.6.26-rc5-mm3/fs/proc/base.c 2008-06-20 13:04:31.000000000 +0200
@@ -2526,7 +2526,7 @@ static const struct pid_entry tgid_base_
  INF("io", S_IRUGO, tgid_io_accounting),
 #endif
 #ifdef CONFIG_SYSVIPC
- REG("semundo",    S_IRUGO, semundo),
+ REG("semundo",    S_IWUSR|S_IRUGO, semundo),
 #endif
 };

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 11:11:21.000000000 +0200
+++ linux-2.6.26-rc5-mm3/include/linux/sem.h 2008-06-20 13:05:14.000000000 +0200
@@ -126,7 +126,8 @@ 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 that changes the list through procfs.
  */
 struct sem_undo_list {
  atomic_t  refcnt;
Index: linux-2.6.26-rc5-mm3/ipc/sem.c
===================================================================
--- linux-2.6.26-rc5-mm3.orig/ipc/sem.c 2008-06-20 12:43:33.000000000 +0200
+++ linux-2.6.26-rc5-mm3/ipc/sem.c 2008-06-20 13:16:56.000000000 +0200
@@ -937,6 +937,12 @@ asmlinkage long sys_semctl (int semid, i
  * This can block, so callers must hold no locks.
  *
  * Note:
+ * task_lock is used to synchronize:
+ *    1. several potential concurrent creations of the undo list.
+ *    2. the undo list removal (upon exit of the task using it). In that case,
+ *       PF_EXITING is checked to avoid creating an undo_list for a task that
+ *       is exiting or has exited.
+ *
  * 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.
@@ -985,7 +991,20 @@ static inline int get_undo_list(struct t
    * 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
+   * 3) semundo_open (procfs write operation):
+   *          means that current task is creating a
```

```
+   *          semundo_list for a target process.
+   *     id as sys_semtimedop
+   *     in that case the target task should not be exiting.
    */
+  if (tsk->flags & PF_EXITING) {
+   /*
+    * Can only happen in the procfs path
+    */
+   task_unlock(tsk);
+   kfree(undo_list);
+   return -EINVAL;
+  }
   atomic_set(&undo_list->refcnt, 2);

   undo_list->ns = get_ipc_ns(tsk->nsproxy->ipc_ns);
@@ -1395,7 +1414,7 @@ static void free_semundo_list(struct sem
  put_ipc_ns(ulp->ns);
  /*
   * We are here if the refcnt became 0, so only a single task can be
-  * accessing that undo_list.
+  * accessing that undo_list: either from exit_sem() or procfs ops.
   */
  kfree(ulp);
 }
@@ -1549,6 +1568,9 @@ 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
+ * the target PID, the semundo list is created here.
  */
 static int semundo_open(struct inode *inode, struct file *file)
 {
@@ -1567,18 +1589,32 @@ 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;
+
+ 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, PROCFS_PATH);
+
+ put_task_struct(task);

- ret = seq_open(file, &semundo_op);
  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))
@@ -1586,6 +1622,233 @@ 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
+}
+
+/*
+ * Reads a line from /proc/<PID>/semundo.
+ * Returns the number of undo values read (or errcode upon failure).
+ * @id: pointer to the semid (filled in with 1st field in the line)
```

```
+ * @array: semundo values (filled in iwth remaining fields in the line).
+ * @array_len: max # of expected semundo values
+ */
+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;
+  /* reached end of line or end of buffer */
+  if (err == INT_MAX)
+   break;
+  /* Return an error if we get more values than expected */
+  if (i < array_len)
+   array[i] = val;
+  else
+   return -EINVAL;
+ }
+ return i;
+}
+
+/*
+ * sets or updates the undo values for the undo_list of 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;
+ int ret = 0;
+
+ un = find_alloc_undo(undo_list, id);
+ if (IS_ERR(un)) {
+  ret = PTR_ERR(un);
+  goto out;
+ }
+
+ /* lookup the sem_array */
```

```
+ sma = sem_lock(ns, id);
+ if (IS_ERR(sma)) {
+  ret = PTR_ERR(sma);
+  rcu_read_unlock();
+  goto out;
+ }
+
+ /*
+  * find_alloc_undo opened an rcu read section to protect un.
+  * Releasing it here is safe:
+  *    . sem_lock is held, so we are protected against IPC_RMID
+  *    . the refcnt won't fall to 0 between semundo_open() and
+  *      semundo_release(), so free_semundo_list won't be called while
+  *      we are here.
+  */
+ rcu_read_unlock();
+
+ /*
+  * semid identifiers are not unique - get_undo_list() (called during
+  * semundo_open()) may have allocated an undo structure, it was
+  * invalidated by an RMID and now a new array received the same id.
+  * Check and fail.
+  * This case can be detected checking un->semid. The existance of
+  * "un" itself is guaranteed by rcu.
+  */
+ if (un->semid == -1) {
+  ret = -EIDRM;
+  goto out_unlock;
+ }
+
+ /*
+  * 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) {
+  ret = -EINVAL;
+  goto out_unlock;
+ }
+
+ /* update the undo values */
+ while (--size >= 0)
+  un->semadj[size] = array[size];
+
+ /* maybe some queued-up processes were waiting for this */
+ update_queue(sma);
+
+out_unlock:
+ sem_unlock(sma);
```

```
+out:
+ return ret;
+}
+
+/*
+ * write operation for /proc/<pid>/semundo file
+ *
+ * The expected string format is:
+ * "<semID> <val1> <val2> ... <valN>"
+ *
+ * It sets (or updates) the sem_undo list for the target <pid> and the target
+ * <semID>, to the given 'undo' values.
+ *
+ * <semID> must match an existing semaphore array.
+ * The number of values following <semID> must match the number of semaphores
+ * in the corresponding array.
+ *
+ * Multiple semID's can be passed simultaneously: newline ('\n') is considered
+ * as a separator in that case.
+ *
+ * Note: it is not allowed to set the sem_undo list for a given semID using
+ *       mutliple write calls.
+ */
+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;
+
+  nval = semundo_readline(&buf, &left, &id, array, max_sem);
```

```
+  if (nval < 0) {
+   err = nval;
+   goto out;
+  }
+
+  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;
@@ -1600,6 +1863,7 @@ 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,
 };

--
```

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