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