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,
 };

--
```