Subject: [PATCH 2.6.24-rc8-mm1 15/15] (RFC) IPC/semaphores: add write() operation to semundo file in procfs
Posted by Pierre Peiffer on Tue, 29 Jan 2008 16:02:44 GMT
View Forum Message <> Reply to Message

From: Pierre Peiffer <pierre.peiffer@bull.net>

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

The user must provide some lines, each containing the semaphores ID
followed by the semaphores values to undo.

The operation failed 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>
---

 fs/proc/base.c | 2
 ipc/sem.c | 232 ++--
 2 files changed, 227 insertions(+), 7 deletions(-)

Index: b/fs/proc/base.c
===
--- a/fs/proc/base.c
+++ b/fs/proc/base.c
@@ -2256,7 +2256,7 @@ static const struct pid_entry tgid_base_
  INF("io", S_IRUGO, pid_io_accounting),
 #endif
 #ifdef CONFIG_SYSVIPC
- REG("semundo",   S_IRUGO, semundo),
+ REG("semundo",   S_IWUSR|S_IRUGO, semundo),
 #endif
 };

Index: b/ipc/sem.c
===
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -1580,6 +1580,9 @@ 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
+ * this target PID, it is created here.

```
 */
 static int semundo_open(struct inode *inode, struct file *file)
 {
@@ -1598,18 +1601,31 @@ 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;

- ret = seq_open(file, &semundo_op);
+ 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);
+
+ put_task_struct(task);
+
  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))
@@ -1617,6 +1633,209 @@ 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
+}
+
+/* semundo_readline: read a line of /proc/<PID>/semundo file
+ * Return the number of value read or an errcode
+ */
+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;
+  /* reach end of line or end of buffer */
+  if (err == INT_MAX)
+   break;
+  /* Return an error if we get more values then expected */
+  if (i < array_len)
+   array[i] = val;
+  else
+   return -EINVAL;
+ }
+ return i;
+}
+
+/* semundo_update: set or update the undo values of the given undo_list
+ * for 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;
+
+retry_undo:
+ un = find_undo(undo_list, id);
+ if (IS_ERR(un))
+  return PTR_ERR(un);
+
+ /* lookup the sem_array */
+ sma = sem_lock(ns, id);
+ if (IS_ERR(sma))
+  return PTR_ERR(sma);
+
+ /*
+  * semid identifiers are not unique - find_undo may have
+  * allocated an undo structure, it was invalidated by an RMID
+  * and now a new array which received the same id.
+  * Check and retry.
+  */
+ if (un->semid == -1) {
+  sem_unlock(sma);
+  goto retry_undo;
+ }
+
+ /*
+  * 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) {
+  sem_unlock(sma);
+  return -EINVAL;
+ }
+
+ /* update the undo values */
+ while (--size >= 0)
+  un->semadj[size] = array[size];
+
+ sem_unlock(sma);
+ return 0;
+}
+
+/*
+ * write operation for /proc/<pid>/semundo file
+ *
+ * It allows to set or update the sem_undo list for this task, for the given
+ * semaphore ID with the given 'undo' values.
```

```
+ *
+ * The expected format of the given string is
+ * "semID <val1> <val2> ... <valN>"
+ *
+ * The semID must match an existing semaphore array and the number of
+ * of values following the semID must match the number of semaphores in
+ * the corresponding array.
+ *
+ * Multiple semID can be passed simultaneously, the newline ('\n') behaving as
+ * a separator in this case.
+ *
+ * Note: passing a sem_undo list (all undo values of one semID) through multiple
+ * write-calls is not allowed.
+ */
+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;
+
+  /* Read a line */
+  nval = semundo_readline(&buf, &left, &id, array, max_sem);
+  if (nval < 0) {
+   err = nval;
+   goto out;
+  }
+
+  /* Update the values for the given semid */
+  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;
@@ -1631,6 +1850,7 @@ 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,
 };

--
Pierre Peiffer
```