

---

Subject: Re: [RFC PATCH 6/6] IPC/sem: .write operation for /proc/<self>/semundo  
Posted by [serue](#) on Wed, 25 Jun 2008 21:09:37 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Quoting Nadia.Derbey@bull.net (Nadia.Derbey@bull.net):

> PATCH [06/06]  
>  
> This patch introduces the .write seq operation for /proc/pid/semundo.  
>  
> In order to simplify the locking strategy, the write operation is only allowed  
> to 'current'.

There should also be a patch against Documentation/filesystems/proc.txt

> Signed-off-by: Nadia Derby <Nadia.Derbey@bull.net>  
>  
> ---  
> fs/proc/base.c | 2  
> ipc/sem.c | 250 ++++++-----  
> 2 files changed, 250 insertions(+), 2 deletions(-)  
>  
> Index: linux-2.6.26-rc5-mm3/fs/proc/base.c  
> =====  
> --- linux-2.6.26-rc5-mm3.orig/fs/proc/base.c 2008-06-24 10:03:33.000000000 +0200  
> +++ linux-2.6.26-rc5-mm3/fs/proc/base.c 2008-06-24 13:21:44.000000000 +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/ipc/sem.c  
> =====  
> --- linux-2.6.26-rc5-mm3.orig/ipc/sem.c 2008-06-24 12:59:15.000000000 +0200  
> +++ linux-2.6.26-rc5-mm3/ipc/sem.c 2008-06-25 08:56:07.000000000 +0200  
> @@ -1554,7 +1554,27 @@ static int semundo\_open(struct inode \*in  
> goto out\_err;  
> }  
>  
> - ulp = get\_proc\_ulp(task);  
> + if (file->f\_flags & O\_WRONLY || file->f\_flags & O\_RDWR) {  
> + /\*  
> + \* Writing is only allowed to current  
> + \*/  
> + if (task != current) {

```

> + put_task_struct(task);
> + ret = -EPERM;
> + goto out_err;
> + }
> + /*
> + * No need to increment the undo_list's refcnt: only current
> + * can be freeing it through exit, and we are current.
> + * Only signal through the ulp NULL pointer that it won't be
> + * necessary to decrement the refcnt during release.
> + * Actually, in this path the undo_list will be gotten during
> + * the write operation.
> + */

```

(I'll probably regret asking this when I turn out to be obviously wrong, but:)

What if I'm opening the file O\_RDWR and first read the file? You'll end up showing no entries, right?

Which may be ok, so long as it's properly documented in proc.txt...

thanks,  
-serge

```

> + ulp = NULL;
> + } else
> + ulp = get_proc_ulp(task);
> +
> ns = get_ipc_ns(task->nsproxy->ipc_ns);
> put_task_struct(task);
>
> @@ -1574,6 +1594,233 @@ out_err:
> 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 BUFLLEN 11
> + int err, neg = 0, left;
> + char s[BUFLLEN], *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 BUFLLEN

```

```

> +}
> +
> +/*
> + * Reads a line from /proc/self/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 with 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.
> + * This is exactly the same code sequence as sys_semtimedop if we have undos.
> + */
> +static inline int semundo_update(struct ipc_namespace *ns, int semid,
> +    short *array, int size)
> +{
> +    struct sem_undo *un;
> +    struct sem_array *sma;
> +    int ret = 0;
> +
> +    un = find_alloc_undo(ns, semid);

```

```

> + if (IS_ERR(un)) {
> +   ret = PTR_ERR(un);
> +   goto out;
> + }
> +
> + /* lookup the sem_array */
> + sma = sem_lock(ns, semid);
> + 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 since exit_sem only operates on
> +  *   . current and we are the current.
> +  */
> + rcu_read_unlock();
> +
> + /*
> +  * semid identifiers are not unique - find_alloc_undo() 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 existence 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/self/semundo file
> + *
> + * Only current is allowed to write to its semundo file.
> + *
> + * The expected string format is:
> + * "<semID> <val1> <val2> ... <valN>"
> + *
> + * It sets (or updates) the sem_undo list for 'current' 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
> + * multiple 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, semid = 0;
> + size_t left = count;
> + struct undo_list_data *data = m->private;
> + struct ipc_namespace *ns = data->ipc_ns;
> +
> + if (data->undo_list)
> + /* Should never happen */
> + return -EINVAL;
> +
> + max_sem = 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, &semid, array, max_sem);
> + if (nval < 0) {
> + err = nval;
> + goto out;
> + }
> +
> + err = semundo_update(ns, semid, 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;
> @@ -1590,6 +1837,7 @@ 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>

---