
Subject: [RFC PATCH 4/5] use next syscall data to change the behavior of IPC_SET

Posted by [Nadia Derby](#) on Tue, 08 Jul 2008 11:24:26 GMT

[View Forum Message](#) <> [Reply to Message](#)

[PATCH 04/05]

This patch uses the value written into the next_syscall_data proc file as a flag to change the way msgctl(IPC_SET), semctl(IPC_SET) and shmctl(IPC_SET) behave.

When "LONG1 1" is echoed to this file, xxxctl(IPC_SET) will set the time fields and the pid fields according to what is specified in the input parameter (while currently only the permission fields are allowed to be set).

The following syscalls are impacted:

- . msgctl(IPC_SET)
- . semctl(IPC_SET)
- . shmctl(IPC_SET)

This makes it easy to restart an ipc object exactly as it was during the checkpoint phase.

Signed-off-by: Nadia Derby <Nadia.Derbey@bull.net>

include/linux/next_syscall_data.h | 12 ++++++++
ipc/msg.c | 19 ++++++++
ipc/sem.c | 16 ++++++++
ipc/shm.c | 19 ++++++++
4 files changed, 63 insertions(+), 3 deletions(-)

Index: linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h

```
=====
--- linux-2.6.26-rc8-mm1.orig/include/linux/next_syscall_data.h 2008-07-08 12:22:47.000000000
+0200
+++ linux-2.6.26-rc8-mm1/include/linux/next_syscall_data.h 2008-07-08 12:24:29.000000000
+0200
@@ -6,6 +6,7 @@
 * . object creation with a predefined id
 * . for a sysv ipc object
 * . for a process
+ * . set more than the usual ipc_perm fields during and IPC_SET operation.
 */
```

```
#ifndef _LINUX_NEXT_SYSCALL_DATA_H
```

```
@@ -20,6 +21,10 @@
```

```
 * by next syscall. The following syscalls support this feature:
 * . msgget(), semget(), shmget()
```

```

    *   . fork(), vfork(), clone()
+ *
+ * If it is set to a non null value before a call to:
+ *   . msgctl(IPC_SET), semctl(IPC_SET), shmctl(IPC_SET),
+ * this means that we are going to set more than the usual ipc_perms fields.
    */
    struct next_syscall_data {
        int ndata;
@@ -35,6 +40,13 @@ struct next_syscall_data {

#define get_next_data(tsk) ((tsk)->nsd->data[0])

+/*
+ * Returns true if next call to xxxctl(IPC_SET) should have a non-default
+ * behavior.
+ */
+#define ipc_set_all(tsk) (next_data_set(tsk) ? get_next_data(tsk) : 0)
+
+
extern ssize_t get_next_syscall_data(struct task_struct *, char *, size_t);
extern int set_next_syscall_data(struct task_struct *, char *);
extern void reset_next_syscall_data(struct task_struct *);
Index: linux-2.6.26-rc8-mm1/ipc/msg.c
=====
--- linux-2.6.26-rc8-mm1.orig/ipc/msg.c 2008-07-08 12:12:36.000000000 +0200
+++ linux-2.6.26-rc8-mm1/ipc/msg.c 2008-07-08 12:26:03.000000000 +0200
@@ -446,7 +446,24 @@ static int msgctl_down(struct ipc_namesp
    msq->q_qbytes = msqid64.msg_qbytes;

    ipc_update_perm(&msqid64.msg_perm, ipcp);
- msq->q_ctime = get_seconds();
+ if (unlikely(ipc_set_all(current))) {
+ /*
+  * If this field is set in the task struct, this
+  * means that we want to set more than the usual
+  * fields. Particularly useful to restart a msgq
+  * in the same state as it was before being
+  * checkpointed.
+  */
+ msq->q_stime = msqid64.msg_stime;
+ msq->q_rtime = msqid64.msg_rtime;
+ msq->q_ctime = msqid64.msg_ctime;
+ msq->q_lspid = msqid64.msg_lspid;
+ msq->q_lrpid = msqid64.msg_lrpid;
+
+ reset_next_syscall_data(current);
+ } else
+ msq->q_ctime = get_seconds();

```

```

+
+ /* sleeping receivers might be excluded by
+  * stricter permissions.
+  */
Index: linux-2.6.26-rc8-mm1/ipc/sem.c
=====
--- linux-2.6.26-rc8-mm1.orig/ipc/sem.c 2008-07-08 12:12:36.000000000 +0200
+++ linux-2.6.26-rc8-mm1/ipc/sem.c 2008-07-08 12:27:06.000000000 +0200
@@ -874,7 +874,21 @@ static int semctl_down(struct ipc_namesp
    goto out_up;
    case IPC_SET:
        ipc_update_perm(&semid64.sem_perm, ipc);
-   sma->sem_ctime = get_seconds();
+
+   if (unlikely(ipc_set_all(current))) {
+       /*
+        * If this field is set in the task struct, this
+        * means that we want to set more than the usual
+        * fields. Particularly useful to restart a semaphore
+        * in the same state as it was before being
+        * checkpointed.
+        */
+       sma->sem_ctime = semid64.sem_ctime;
+       sma->sem_otime = semid64.sem_otime;
+
+       reset_next_syscall_data(current);
+   } else
+       sma->sem_ctime = get_seconds();
        break;
    default:
        err = -EINVAL;
Index: linux-2.6.26-rc8-mm1/ipc/shm.c
=====
--- linux-2.6.26-rc8-mm1.orig/ipc/shm.c 2008-07-08 12:12:36.000000000 +0200
+++ linux-2.6.26-rc8-mm1/ipc/shm.c 2008-07-08 12:27:32.000000000 +0200
@@ -609,7 +609,24 @@ static int shmctl_down(struct ipc_namesp
    goto out_up;
    case IPC_SET:
        ipc_update_perm(&shmid64.shm_perm, ipc);
-   shp->shm_ctim = get_seconds();
+
+   if (unlikely(ipc_set_all(current))) {
+       /*
+        * If this field is set in the task struct, this
+        * means that we want to set more than the usual
+        * fields. Particularly useful to restart a shm seg
+        * in the same state as it was before being
+        * checkpointed.

```

```
+ */
+ shp->shm_atim = shmid64.shm_atime;
+ shp->shm_dtim = shmid64.shm_dtime;
+ shp->shm_ctim = shmid64.shm_ctime;
+ shp->shm_cprid = shmid64.shm_cpid;
+ shp->shm_lprid = shmid64.shm_lpid;
+
+ reset_next_syscall_data(current);
+ } else
+ shp->shm_ctim = get_seconds();
+ break;
+ default:
+ err = -EINVAL;

--
```

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