
Subject: [RFC][PATCH 1/4] Provide a new procfs interface to set next id
Posted by [Nadia Derby](#) on Fri, 04 Apr 2008 14:51:30 GMT
[View Forum Message](#) <> [Reply to Message](#)

[PATCH 01/04]

This patch proposes the procfs facilities needed to feed the id for the next object to be allocated.

if an
echo "LONG XX" > /proc/self/next_id
is issued, next object to be created will have XX as its id.

This applies to objects that need a single id, such as ipc objects.

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

fs/exec.c		3	+
fs/proc/base.c		73	+++++
include/linux/sched.h		3	+
include/linux/sysids.h		24	+++++
kernel/Makefile		2	-
kernel/exit.c		4	++
kernel/fork.c		2	+
kernel/nextid.c		86	+++++

8 files changed, 196 insertions(+), 1 deletion(-)

Index: linux-2.6.25-rc8-mm1/include/linux/sysids.h

=====

--- /dev/null 1970-01-01 00:00:00.000000000 +0000

+++ linux-2.6.25-rc8-mm1/include/linux/sysids.h 2008-04-04 13:53:04.000000000 +0200

@@ -0,0 +1,24 @@

+/*

+ * include/linux/sysids.h

+ *

+ * Definitions to support object creation with predefined id.

+ *

+ */

+

+#ifndef _LINUX_SYSIDS_H

+#define _LINUX_SYSIDS_H

+

+struct sys_id {

+ long id;

+};

+

+extern ssize_t get_nextid(struct task_struct *, char *, size_t);


```

+
+ if (count >= PAGE_SIZE)
+   count = PAGE_SIZE - 1;
+
+ length = -ENOMEM;
+ page = (char *) __get_free_page(GFP_TEMPORARY);
+ if (!page)
+   goto out;
+
+ length = get_nextid(task, (char *) page, count);
+ if (length >= 0)
+   length = simple_read_from_buffer(buf, count, ppos,
+   (char *)page, length);
+ free_page((unsigned long) page);
+
+out:
+ put_task_struct(task);
+ return length;
+}
+
+static ssize_t next_id_write(struct file *file, const char __user *buf,
+   size_t count, loff_t *ppos)
+{
+ struct inode *inode = file->f_path.dentry->d_inode;
+ char *page;
+ ssize_t length;
+
+ if (pid_task(proc_pid(inode), PIDTYPE_PID) != current)
+   return -EPERM;
+
+ if (count >= PAGE_SIZE)
+   count = PAGE_SIZE - 1;
+
+ if (*ppos != 0) {
+   /* No partial writes. */
+   return -EINVAL;
+ }
+ page = (char *) __get_free_page(GFP_TEMPORARY);
+ if (!page)
+   return -ENOMEM;
+ length = -EFAULT;
+ if (copy_from_user(page, buf, count))
+   goto out_free_page;
+
+ page[count] = '\0';
+
+ length = set_nextid(current, page);
+ if (!length)

```



```

+++ linux-2.6.25-rc8-mm1/kernel/nextid.c 2008-04-04 13:59:59.000000000 +0200
@@ -0,0 +1,86 @@
+/*
+ * linux/kernel/nextid.c
+ *
+ * Provide the get_nextid() / set_nextid() routines
+ * (called from fs/proc/base.c).
+ * They allow to specify the id for the next resource to be allocated,
+ * instead of letting the allocator set it for us.
+ */
+
+#include <linux/sched.h>
+#include <linux/ctype.h>
+
+
+
+ssize_t get_nextid(struct task_struct *task, char *buffer, size_t size)
+{
+ struct sys_id *sid;
+
+ sid = task->next_id;
+ if (!sid)
+ return snprintf(buffer, size, "UNSET\n");
+
+ return snprintf(buffer, size, "LONG %ld\n", sid->id);
+}
+
+static int set_single_id(struct task_struct *task, char *buffer)
+{
+ struct sys_id *sid;
+ long next_id;
+ char *end;
+
+ next_id = simple_strtol(buffer, &end, 0);
+ if (end == buffer || (end && !isspace(*end)))
+ return -EINVAL;
+
+ sid = task->next_id;
+ if (!sid) {
+ sid = kzalloc(sizeof(*sid), GFP_KERNEL);
+ if (!sid)
+ return -ENOMEM;
+ task->next_id = sid;
+ }
+
+ sid->id = next_id;
+
+

```

```

+ return 0;
+}
+
+int reset_nextid(struct task_struct *task)
+{
+ struct sys_id *sid;
+
+ sid = task->next_id;
+ if (!sid)
+ return 0;
+
+ task->next_id = NULL;
+ kfree(sid);
+ return 0;
+}
+
+#define LONG_STR "LONG"
+#define RESET_STR "RESET"
+
+/*
+ * Parses a line written to /proc/self/next_id.
+ * this line has the following format:
+ * LONG id --> a single id is specified
+ */
+int set_nextid(struct task_struct *task, char *buffer)
+{
+ char *token, *out = buffer;
+
+ if (!out)
+ return -EINVAL;
+
+ token = strsep(&out, " ");
+
+ if (!strcmp(token, LONG_STR))
+ return set_single_id(task, out);
+ else if (!strncmp(token, RESET_STR, strlen(RESET_STR)))
+ return reset_nextid(task);
+ else
+ return -EINVAL;
+}

```

Index: linux-2.6.25-rc8-mm1/kernel/fork.c

```

=====
--- linux-2.6.25-rc8-mm1.orig/kernel/fork.c 2008-04-04 13:11:37.000000000 +0200
+++ linux-2.6.25-rc8-mm1/kernel/fork.c 2008-04-04 14:00:35.000000000 +0200
@@ -1167,6 +1167,8 @@ static struct task_struct *copy_process(
    p->blocked_on = NULL; /* not blocked yet */
#endif

```

```
+ p->next_id = NULL;
+
+ /* Perform scheduler related setup. Assign this task to a CPU. */
+ sched_fork(p, clone_flags);
```

Index: linux-2.6.25-rc8-mm1/kernel/exit.c

```
=====
--- linux-2.6.25-rc8-mm1.orig/kernel/exit.c 2008-04-04 13:11:37.000000000 +0200
+++ linux-2.6.25-rc8-mm1/kernel/exit.c 2008-04-04 14:01:22.000000000 +0200
@@ -987,6 +987,10 @@ NORET_TYPE void do_exit(long code)
```

```
    proc_exit_connector(tsk);
    exit_notify(tsk, group_dead);
```

```
+
+ if (unlikely(tsk->next_id))
+   exit_nextid(tsk);
+
```

```
#ifdef CONFIG_NUMA
    mpol_free(tsk->mempolicy);
    tsk->mempolicy = NULL;
```

Index: linux-2.6.25-rc8-mm1/fs/exec.c

```
=====
--- linux-2.6.25-rc8-mm1.orig/fs/exec.c 2008-04-04 13:11:34.000000000 +0200
+++ linux-2.6.25-rc8-mm1/fs/exec.c 2008-04-04 14:02:09.000000000 +0200
@@ -1024,6 +1024,9 @@ int flush_old_exec(struct linux_binprm *
    flush_signal_handlers(current, 0);
    flush_old_files(current->files);
```

```
+ if (unlikely(current->next_id))
+   reset_nextid(current);
+
+   return 0;
```

mmap_failed:

--

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