

---

Subject: [RFC][PATCH 1/4] Provide a new procfs interface to set next id  
Posted by Nadia Derbey on Fri, 28 Mar 2008 09:53:10 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 Derbey <[Nadia.Derbey@bull.net](mailto:Nadia.Derbey@bull.net)>

---

```
fs/proc/base.c      |  73 ++++++++++++++++++++++++++++++++++++++++
include/linux/sched.h |   3 ++
include/linux/sysids.h |  18 ++++++
kernel/Makefile      |   2 -
kernel/fork.c        |   2 +
kernel/nextid.c      |  69 ++++++
6 files changed, 166 insertions(+), 1 deletion(-)
```

Index: linux-2.6.25-rc3-mm1/include/linux/sysids.h

```
--- /dev/null 1970-01-01 00:00:00.000000000 +0000
+++ linux-2.6.25-rc3-mm1/include/linux/sysids.h 2008-03-27 18:02:08.000000000 +0100
@@ -0,0 +1,18 @@
+/*
+ * 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 *);
+extern int set_nextid(struct task_struct *, char *);
+
```

```

+#endif /* _LINUX_SYSIDS_H */
Index: linux-2.6.25-rc3-mm1/include/linux/sched.h
=====
--- linux-2.6.25-rc3-mm1.orig/include/linux/sched.h 2008-03-10 09:18:46.000000000 +0100
+++ linux-2.6.25-rc3-mm1/include/linux/sched.h 2008-03-10 09:28:30.000000000 +0100
@@ -87,6 +87,7 @@ struct sched_param {
#include <linux/task_io_accounting.h>
#include <linux/kobject.h>
#include <linux/latencytop.h>
+#include <linux/sysids.h>

#include <asm/processor.h>

@@ -1261,6 +1262,8 @@ struct task_struct {
    int latency_record_count;
    struct latency_record latency_record[LT_SAVECOUNT];
#endif
+ /* Id to assign to the next resource to be created */
+ struct sys_id *next_id;
};

/*
Index: linux-2.6.25-rc3-mm1/fs/proc/base.c
=====
--- linux-2.6.25-rc3-mm1.orig/fs/proc/base.c 2008-03-10 09:19:39.000000000 +0100
+++ linux-2.6.25-rc3-mm1/fs/proc/base.c 2008-03-21 12:03:09.000000000 +0100
@@ -1080,6 +1080,77 @@ static const struct file_operations proc
#endif

+static ssize_t next_id_read(struct file *file, char __user *buf,
+   size_t count, loff_t *ppos)
+{
+ struct task_struct *task;
+ char *page;
+ ssize_t length;
+
+ task = get_proc_task(file->f_path.dentry->d_inode);
+ if (!task)
+ return -ESRCH;
+
+ if (count > PROC_BLOCK_SIZE)
+ count = PROC_BLOCK_SIZE;
+
+ length = -ENOMEM;
+ page = (char *) __get_free_page(GFP_TEMPORARY);
+ if (!page)
+ goto out;
+

```

```

+
+ length = get_nextid(task, (char *) page);
+ 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)
+   length = count;
+
+out_free_page:
+ free_page((unsigned long) page);
+ return length;
+}
+
+static const struct file_operations proc_next_id_operations = {

```

```

+ .read = next_id_read,
+ .write = next_id_write,
+};
+
+
#ifndef CONFIG_SCHED_DEBUG
/*
 * Print out various scheduling related per-task fields:
@@ -2391,6 +2462,7 @@ static const struct pid_entry tgid_base_
#endif CONFIG_TASK_IO_ACCOUNTING
INF("io", S_IRUGO, pid_io_accounting),
#endif
+ REG("next_id", S_IRUGO|S_IWUSR, next_id),
};

static int proc_tgid_base_readdir(struct file * filp,
@@ -2716,6 +2788,7 @@ static const struct pid_entry tid_base_s
#endif CONFIG_FAULT_INJECTION
REG("make-it-fail", S_IRUGO|S_IWUSR, fault_inject),
#endif
+ REG("next_id", S_IRUGO|S_IWUSR, next_id),
};

static int proc_tid_base_readdir(struct file * filp,
Index: linux-2.6.25-rc3-mm1/kernel/Makefile
=====
--- linux-2.6.25-rc3-mm1.orig/kernel/Makefile 2008-03-10 09:19:01.000000000 +0100
+++ linux-2.6.25-rc3-mm1/kernel/Makefile 2008-03-20 17:29:50.000000000 +0100
@@ -9,7 +9,7 @@ obj-y = sched.o fork.o exec_domain.o
        rcupdate.o extable.o params.o posix-timers.o \
        kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
        hrtimer.o rwsem.o nsproxy.o srcu.o semaphore.o \
-       notifier.o ksysfs.o pm_qos_params.o
+       notifier.o ksysfs.o pm_qos_params.o nextid.o

obj-$(CONFIG_SYSCTL_SYSCALL_CHECK) += sysctl_check.o
obj-$(CONFIG_STACKTRACE) += stacktrace.o
Index: linux-2.6.25-rc3-mm1/kernel/nextid.c
=====
--- /dev/null 1970-01-01 00:00:00.000000000 +0000
+++ linux-2.6.25-rc3-mm1/kernel/nextid.c 2008-03-27 18:02:08.000000000 +0100
@@ -0,0 +1,69 @@
+/*
+ * 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)
+{
+ struct sys_id *sid;
+
+ sid = task->next_id;
+ if (!sid)
+ return snprintf(buffer, sizeof(buffer), "-1\n");
+
+ return snprintf(buffer, sizeof(buffer), "%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;
+}
+
+#define SINGLE_LONG "LONG"
+
+/*
+ * 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;
+
+ token = strsep(&out, " ");
+ if (!token || !out)
+ return -EINVAL;
+
+ if (!strcmp(token, SINGLE_LONG))
+ return set_single_id(task, out);
+ else
+ return -EINVAL;
+}
```

Index: linux-2.6.25-rc3-mm1/kernel/fork.c

```
=====
--- linux-2.6.25-rc3-mm1.orig/kernel/fork.c 2008-03-27 18:01:56.000000000 +0100
+++ linux-2.6.25-rc3-mm1/kernel/fork.c 2008-03-27 18:02:42.000000000 +0100
@@ -1166,6 +1166,8 @@ 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);
```

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