
Subject: [PATCH 2/4] - v2 - Provide a new procfs interface to set next upid nr(s)
Posted by [Nadia Derby](#) on Fri, 18 Apr 2008 05:45:01 GMT
[View Forum Message](#) <> [Reply to Message](#)

[PATCH 02/04]

This patch proposes the procfs facilities needed to feed the id(s) for the next task to be forked.

say n is the number of pids to be provided through procfs:

if an
echo "LONG<n> X0 X1 ... X<n-1>" > /proc/self/task/<tid>/next_id
is issued, the next task to be forked will have its upid nrs set as follows
(say it is forked in a pid ns of level L):

```
level      upid nr
L -----> X0
..
L - i -----> Xi
..
L - n + 1 --> X<n-1>
```

Then, for levels L-n down to level 0, the pids will be left to the kernel choice.

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

```
---
include/linux/sysids.h | 27 ++++++++
kernel/nextid.c        | 158 ++++++++++++++++++++++++++++++++++++++-----
2 files changed, 163 insertions(+), 22 deletions(-)
```

Index: linux-2.6.25-rc8-mm2/include/linux/sysids.h

```
=====
--- linux-2.6.25-rc8-mm2.orig/include/linux/sysids.h 2008-04-17 13:35:29.000000000 +0200
+++ linux-2.6.25-rc8-mm2/include/linux/sysids.h 2008-04-17 16:03:07.000000000 +0200
@@ -8,8 +8,33 @@
 #ifndef _LINUX_SYSIDS_H
 #define _LINUX_SYSIDS_H

+
+#define NIDS_SMALL      32
+#define NIDS_PER_BLOCK ((unsigned int)(PAGE_SIZE / sizeof(long)))
+
+/* access the ids "array" with this macro */
+#define ID_AT(pi, i) \
+ ((pi)->blocks[(i) / NIDS_PER_BLOCK][(i) % NIDS_PER_BLOCK])
```

```

+
+
+/*
+ * List of ids for the next object to be created. This presently applies to
+ * next process to be created.
+ * The next process to be created is associated to a set of upid nrs: one for
+ * each pid namespace level that process belongs to.
+ * upid nrs from level 0 up to level <nids - 1> will be automatically
+ * allocated.
+ * upid nr for level nids will be set to blocks[0][0]
+ * upid nr for level <nids + i> will be set to ID_AT(ids, i);
+ *
+ * If a single id is needed, nids is set to 1 and small_block[0] is set to
+ * that id.
+ */

```

```

    struct sys_id {
- long id;
+ int nids;
+ long small_block[NIDS_SMALL];
+ int nblocks;
+ long *blocks[0];
    };

```

```

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

```

```

Index: linux-2.6.25-rc8-mm2/kernel/nextid.c

```

```

=====
--- linux-2.6.25-rc8-mm2.orig/kernel/nextid.c 2008-04-17 15:16:07.000000000 +0200
+++ linux-2.6.25-rc8-mm2/kernel/nextid.c 2008-04-17 16:54:30.000000000 +0200
@@ -13,38 +13,146 @@

```

```

+static struct sys_id *id_blocks_alloc(int nids)
+{
+ struct sys_id *ids;
+ int nblocks;
+ int i;
+
+ nblocks = (nids + NIDS_PER_BLOCK - 1) / NIDS_PER_BLOCK;
+ BUG_ON(nblocks < 1);
+
+ ids = kmalloc(sizeof(*ids) + nblocks * sizeof(long *), GFP_KERNEL);
+ if (!ids)
+ return NULL;
+ ids->nids = nids;
+ ids->nblocks = nblocks;
+
+ if (nids <= NIDS_SMALL)

```

```

+ ids->blocks[0] = ids->small_block;
+ else {
+   for (i = 0; i < nblocks; i++) {
+     long *b;
+     b = (void *)__get_free_page(GFP_KERNEL);
+     if (!b)
+       goto out_undo_partial_alloc;
+     ids->blocks[i] = b;
+   }
+ }
+ return ids;
+
+out_undo_partial_alloc:
+ while (--i >= 0)
+   free_page((unsigned long)ids->blocks[i]);
+
+ kfree(ids);
+ return NULL;
+}
+
+static void id_blocks_free(struct sys_id *ids)
+{
+ if (ids == NULL)
+   return;
+
+ if (ids->blocks[0] != ids->small_block) {
+   int i;
+   for (i = 0; i < ids->nblocks; i++)
+     free_page((unsigned long)ids->blocks[i]);
+ }
+ kfree(ids);
+ return;
+}
+
+ssize_t get_nextid(struct task_struct *task, char *buffer, size_t size)
+{
+   ssize_t rc, count = 0;
+   struct sys_id *sid;
+   char *bufptr = buffer;
+   int i;

+   sid = task->next_id;
+   if (!sid)
+     return 0;
+   if (!sid || !sid->nids)
+     return snprintf(buffer, size, "UNSET\n");

+   return snprintf(buffer, size, "LONG %ld\n", sid->id);
+   count = snprintf(bufptr, size, "LONGS (%d) ", sid->nids);

```

```

+
+ for (i = 0; i < sid->nids - 1; i++) {
+   rc = snprintf(&bufptr[count], size - count, "%ld ",
+     ID_AT(sid, i));
+   if (rc >= size - count)
+     return -ENOMEM;
+   count += rc;
+ }
+
+ rc = snprintf(&bufptr[count], size - count, "%ld\n", ID_AT(sid, i));
+ if (rc >= size - count)
+   return -ENOMEM;
+ count += rc;
+
+ return count;
+ }

-static int set_single_id(struct task_struct *task, char *buffer)
+static int fill_nextid_list(struct task_struct *task, int nids, char *buffer)
+ {
- struct sys_id *sid;
- long next_id;
+ char *token, *buff = buffer;
+   char *end;
+ struct sys_id *sid;
+ struct sys_id *old_list = task->next_id;
+ int i;

- next_id = simple_strtol(buffer, &end, 0);
- if (end == buffer || (end && *end && !isspace(*end)))
-   return -EINVAL;
+ sid = id_blocks_alloc(nids);
+ if (!sid)
+   return -ENOMEM;

- sid = task->next_id;
- if (!sid) {
-   sid = kzalloc(sizeof(*sid), GFP_KERNEL);
-   if (!sid)
-     return -ENOMEM;
-   task->next_id = sid;
+ i = 0;
+ while ((token = strsep(&buff, " ")) != NULL && i < nids) {
+   long id;
+
+   if (!*token)
+     goto out_free;
+   id = simple_strtol(token, &end, 0);

```

```

+ if (end == token || (*end && !isspace(*end)))
+ goto out_free;
+ ID_AT(sid, i) = id;
+ i++;
+ }

- sid->id = next_id;
+ if (i != nids)
+ /* Not enough pids compared to npids */
+ goto out_free;
+
+ if (old_list)
+ id_blocks_free(old_list);

+ task->next_id = sid;
+ return 0;
+
+out_free:
+ id_blocks_free(sid);
+ return -EINVAL;
+}
+
+/*
+ * Parses a line with the following format:
+ * <x> <id0> ... <idx-1>
+ * and sets <id0> to <idx-1> as the sequence of ids to be used for the next
+ * object to be created by the task.
+ * This applies to processes that need 1 id per namespace level.
+ * Any trailing character on the line is skipped.
+ */
+static int set_multiple_ids(struct task_struct *task, char *nb, char *buffer)
+{
+ int nids;
+ char *end;
+
+ nids = simple_strtol(nb, &end, 0);
+ if (*end)
+ return -EINVAL;
+
+ if (nids <= 0)
+ return -EINVAL;
+
+ return fill_nextid_list(task, nids, buffer);
+}

int reset_nextid(struct task_struct *task)
@@ -55,8 +163,8 @@ int reset_nextid(struct task_struct *tas
if (!sid)

```

```

    return 0;

+ id_blocks_free(sid);
  task->next_id = NULL;
- kfree(sid);
  return 0;
}

@@ -65,12 +173,14 @@ int reset_nextid(struct task_struct *tas

/*
 * Parses a line written to /proc/self/task/<my_tid>/next_id.
- * this line has the following format:
+ * this line has one of the following formats:
  * LONG id          --> a single id is specified
+ * LONG<x> id0 ... id<x-1> --> a sequence of ids is specified
 */
int set_nextid(struct task_struct *task, char *buffer)
{
  char *token, *out = buffer;
+ size_t sz;

  if (!out)
    return -EINVAL;
@@ -78,9 +188,15 @@ int set_nextid(struct task_struct *task,
  token = strsep(&out, " ");

  if (!strcmp(token, LONG_STR))
- return set_single_id(task, out);
- else if (!strncmp(token, RESET_STR, strlen(RESET_STR)))
+ return fill_nextid_list(task, 1, out);
+
+ sz = strlen(LONG_STR);
+
+ if (!strncmp(token, LONG_STR, sz))
+ return set_multiple_ids(task, token + sz, out);
+
+ if (!strncmp(token, RESET_STR, strlen(RESET_STR)))
  return reset_nextid(task);
- else
- return -EINVAL;
+
+ return -EINVAL;
}

--

```

Containers mailing list

