Subject: [RFC][PATCH 2/4] Provide a new procfs interface to set next upid nr(s)
Posted by Nadia Derbey on Fri, 28 Mar 2008 09:53:11 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/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 Derbey <Nadia.Derbey@bull.net>

---
include/linux/sysids.h | 27 ++++++++-
kernel/nextid.c | 146 ++-------
2 files changed, 153 insertions(+), 20 deletions(-)

Index: linux-2.6.25-rc3-mm1/include/linux/sysids.h
===
--- linux-2.6.25-rc3-mm1.orig/include/linux/sysids.h 2008-03-27 18:02:08.000000000 +0100
+++ linux-2.6.25-rc3-mm1/include/linux/sysids.h 2008-03-28 08:19:49.000000000 +0100
@@ -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 <npids - 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 *);
Index: linux-2.6.25-rc3-mm1/kernel/nextid.c
===================================================================
--- linux-2.6.25-rc3-mm1.orig/kernel/nextid.c 2008-03-27 18:02:08.000000000 +0100
+++ linux-2.6.25-rc3-mm1/kernel/nextid.c 2008-03-28 08:20:52.000000000 +0100
@@ -13,46 +13,148 @@



+static struct sys_id *id_blocks_alloc(int idsetsize)
+{
+ struct sys_id *ids;
+ int nblocks;
+ int i;
+
+ nblocks = (idsetsize + 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 = idsetsize;
+ ids->nblocks = nblocks;
+
+ if (idsetsize <= 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]);
+ }
+ ids->nids = 0;
+ return;
+}
+
 ssize_t get_nextid(struct task_struct *task, char *buffer)
 {
+ ssize_t count = 0;
  struct sys_id *sid;
+ char *bufptr = buffer;
+ int i;

  sid = task->next_id;
- if (!sid)
+ if (!sid || !sid->nids)
   return snprintf(buffer, sizeof(buffer), "-1\n");

- return snprintf(buffer, sizeof(buffer), "%ld\n", sid->id);
+ for (i = 0; i < sid->nids - 1; i++)
```

```
+  count += sprintf(&bufptr[count], "%ld ", ID_AT(sid, i));
+
+ count += sprintf(&bufptr[count], "%ld", ID_AT(sid, i));
+
+ 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 && !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++;
+ }
+
+ if (i != nids)
+  /* Not enough pids compared to npids */
+  goto out_free;
+
```

```
+ if (old_list) {
+  id_blocks_free(old_list);
+  kfree(old_list);
  }

- sid->id = next_id;
+ 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);
 }

 #define SINGLE_LONG "LONG"

 /*
  * Parses a line written to /proc/self/next_id.
- * this line has the following format:
+ * this line has one of the following format:
  * 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)
```

```
 {
@@ -63,7 +165,13 @@ int set_nextid(struct task_struct *task,
   return -EINVAL;

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

--
```