Subject: [RFC][PATCH 2/4] Provide a new procfs interface to set next upid nr(s)
Posted by Nadia Derbey on Mon, 10 Mar 2008 13:50:56 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 "n X0 X1 ... X<n-1>" > /proc/self/next_pids
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>

---
fs/proc/base.c | 74 ++++++++++++++++++++++++
include/linux/sysids.h | 36 ++++++++++-
kernel/set_nextid.c | 147 ++-
3 files changed, 254 insertions(+), 3 deletions(-)

Index: linux-2.6.25-rc3-mm1/include/linux/sysids.h
===
--- linux-2.6.25-rc3-mm1.orig/include/linux/sysids.h 2008-03-10 11:39:10.000000000 +0100
+++ linux-2.6.25-rc3-mm1/include/linux/sysids.h 2008-03-10 12:49:27.000000000 +0100
@@ -9,12 +9,46 @@
#define _LINUX_SYSIDS_H

#define SYS_ID_IPC 1
+#define SYS_ID_PID 2
+
+#define NPIDS_SMALL 32
+#define NPIDS_PER_BLOCK ((unsigned int)(PAGE_SIZE / sizeof(pid_t)))
+
+/* access the pids "array" with this macro */

```
+#define PID_AT(pi, i) \
+ ((pi)->blocks[(i) / NPIDS_PER_BLOCK][(i) % NPIDS_PER_BLOCK])
+
+
+/*
+ * 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 npids will be set to blocks[0][0]
+ * upid nr for level <npids + i> will be set to PID_AT(pids, i);
+ */
+struct pid_list {
+ int npids;
+ pid_t small_block[NPIDS_SMALL];
+ int nblocks;
+ pid_t *blocks[0];
+};
+

 struct sys_id {
  int flag; /* which id should be set */
- int ipc;
+ struct {
+  int ipc;
+  struct pid_list *pids;
+ } ids;
 };

+#define ipc_id ids.ipc
+#define pid_ids ids.pids
+
+extern void pids_free(struct pid_list *);
 extern int ipc_set_nextid(struct task_struct *, int id);
+extern ssize_t pid_get_nextids(struct task_struct *, char *);
+extern ssize_t pid_set_nextids(struct task_struct *, char *);

 #endif /* _LINUX_SYSIDS_H */
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 11:22:20.000000000 +0100
+++ linux-2.6.25-rc3-mm1/fs/proc/base.c 2008-03-10 12:27:34.000000000 +0100
@@ -1095,7 +1095,7 @@ static ssize_t next_ipcid_read(struct fi
   return -ESRCH;

  sid = task->next_id;
- next_ipcid = (sid) ? ((sid->flag & SYS_ID_IPC) ? sid->ipc : -1)
+ next_ipcid = (sid) ? ((sid->flag & SYS_ID_IPC) ? sid->ipc_id : -1)
```

```
    : -1;

  put_task_struct(task);
@@ -1144,6 +1144,76 @@ static const struct file_operations proc
 };


+static ssize_t next_pids_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 = pid_get_nextids(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_pids_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 (current != pid_task(proc_pid(inode), PIDTYPE_PID))
+  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 = pid_set_nextids(current, page);
+ if (!length)
+  length = count;
+
+out_free_page:
+ free_page((unsigned long) page);
+ return length;
+}
+
+static const struct file_operations proc_next_pids_operations = {
+ .read  = next_pids_read,
+ .write  = next_pids_write,
+};
+
+
 #ifdef CONFIG_SCHED_DEBUG
 /*
  * Print out various scheduling related per-task fields:
@@ -2456,6 +2526,7 @@ static const struct pid_entry tgid_base_
  INF("io", S_IRUGO, pid_io_accounting),
 #endif
  REG("next_ipcid", S_IRUGO|S_IWUSR, next_ipcid),
+ REG("next_pids",  S_IRUGO|S_IWUSR, next_pids),
 };

 static int proc_tgid_base_readdir(struct file * filp,
@@ -2782,6 +2853,7 @@ static const struct pid_entry tid_base_s
  REG("make-it-fail", S_IRUGO|S_IWUSR, fault_inject),
 #endif
  REG("next_ipcid", S_IRUGO|S_IWUSR, next_ipcid),
+ REG("next_pids",  S_IRUGO|S_IWUSR, next_pids),
 };

 static int proc_tid_base_readdir(struct file * filp,
```

```
Index: linux-2.6.25-rc3-mm1/kernel/set_nextid.c
===================================================================
--- linux-2.6.25-rc3-mm1.orig/kernel/set_nextid.c 2008-03-10 10:09:47.000000000 +0100
+++ linux-2.6.25-rc3-mm1/kernel/set_nextid.c 2008-03-10 12:47:30.000000000 +0100
@@ -8,8 +8,59 @@
  */

 #include <linux/sched.h>
+#include <linux/string.h>


+extern int pid_max;
+
+
+
+static struct pid_list *pids_alloc(int idsetsize)
+{
+ struct pid_list *pids;
+ int nblocks;
+ int i;
+
+ nblocks = (idsetsize + NPIDS_PER_BLOCK - 1) / NPIDS_PER_BLOCK;
+ BUG_ON(nblocks < 1);
+
+ pids = kmalloc(sizeof(*pids) + nblocks * sizeof(pid_t *), GFP_KERNEL);
+ if (!pids)
+  return NULL;
+ pids->npids = idsetsize;
+ pids->nblocks = nblocks;
+
+ if (idsetsize <= NPIDS_SMALL)
+  pids->blocks[0] = pids->small_block;
+ else {
+  for (i = 0; i < nblocks; i++) {
+   pid_t *b;
+   b = (void *)__get_free_page(GFP_KERNEL);
+   if (!b)
+    goto out_undo_partial_alloc;
+   pids->blocks[i] = b;
+  }
+ }
+ return pids;
+
+out_undo_partial_alloc:
+ while (--i >= 0)
+  free_page((unsigned long)pids->blocks[i]);
+
+ kfree(pids);
```

```
+ return NULL;
+}
+
+void pids_free(struct pid_list *pids)
+{
+ if (pids->blocks[0] != pids->small_block) {
+  int i;
+  for (i = 0; i < pids->nblocks; i++)
+   free_page((unsigned long)pids->blocks[i]);
+ }
+ kfree(pids);
+}
+

 int ipc_set_nextid(struct task_struct *task, int id)
 {
@@ -23,9 +74,103 @@ int ipc_set_nextid(struct task_struct *t
   task->next_id = sid;
  }

- sid->ipc = id;
+ sid->ipc_id = id;
  sid->flag |= SYS_ID_IPC;

  return 0;
 }

+ssize_t pid_get_nextids(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)
+  return sprintf(buffer, "-1");
+
+ if (!(sid->flag & SYS_ID_PID))
+  return sprintf(buffer, "-1");
+
+ count = sprintf(&bufptr[count], "%d ", sid->pid_ids->npids);
+
+ for (i = 0; i < sid->pid_ids->npids - 1; i++)
+  count += sprintf(&bufptr[count], "%d ",
+    PID_AT(sid->pid_ids, i));
+
+ count += sprintf(&bufptr[count], "%d", PID_AT(sid->pid_ids, i));
```

```
+
+ return count;
+}
+
+/*
+ * Parses a line written to /proc/self/next_pids.
+ * this line has the following format:
+ * npids pid0  .... pidx
+ * with x = npids - 1
+ */
+ssize_t pid_set_nextids(struct task_struct *task, char *buffer)
+{
+ char *token, *end, *out = buffer;
+ struct sys_id *sid;
+ struct pid_list *pids;
+ int npids, i;
+ ssize_t rc;
+
+ rc = -EINVAL;
+ token = strsep(&out, " ");
+ if (!token)
+  goto out;
+
+ npids = simple_strtol(token, &end, 0);
+ if (*end)
+  goto out;
+
+ if (npids <= 0 || npids > pid_max)
+  goto out;
+
+ rc = -ENOMEM;
+ pids = pids_alloc(npids);
+ if (!pids)
+  goto out;
+
+ rc = -EINVAL;
+ i = 0;
+ while ((token = strsep(&out, " ")) != NULL && i < npids) {
+  pid_t pid;
+
+  if (!*token)
+   goto out_free;
+  pid = simple_strtol(token, &end, 0);
+  if ((*end && *end != '\n') || end == token || pid < 0)
+   goto out_free;
+  PID_AT(pids, i) = pid;
+  i++;
+ }
```

```
+
+ if (i != npids)
+  /* Not enough pids compared to npids */
+  goto out_free;
+
+ sid = current->next_id;
+ if (!sid) {
+  rc = -ENOMEM;
+  sid = kzalloc(sizeof(*sid), GFP_KERNEL);
+  if (!sid)
+   goto out_free;
+  current->next_id = sid;
+ } else if (sid->flag & SYS_ID_PID)
+  kfree(sid->pid_ids);
+
+ rc = 0;
+
+ sid->pid_ids = pids;
+ sid->flag |= SYS_ID_PID;
+out:
+ return rc;
+out_free:
+ pids_free(pids);
+ return rc;
+}

--
```