Subject: [RFC][PATCH 0/4] Object creation with a pre-defined id (v2)
Posted by Nadia Derbey on Fri, 28 Mar 2008 09:53:09 GMT
View Forum Message <> Reply to Message

Hi,

Here is a second version of what has been proposed 2 weeks ago to create
an object with a pre-defined id (this feature would be used during the
restart operation) - see thread
https://lists.linux-foundation.org/pipermail/containers/2008-March/thread.html#10287

Main changes since last version:
. Pavel's suggestion has been integrated; this makes things more readable:
  alloc_pidmap() is unchanged and a alloc_fixed_pidmap() is added for the
  predefined ids.
. Oren's suggestion has been integrated:
  We now have a single file under /proc/self (/proc/self/next_id).
  When this file is filled, a structure pointed to by the calling task struct
  is filled with the id(s).
  Then, when the object is created, the id(s) present in that structure are
  used, instead of the default ones.
  The syntax is one of:
  . echo "LONG XX" > /proc/self/next_id
    next object to be created will have an id set to XX
  . echo "LONG<n> X0 ... X<n-1>" > /proc/self/next_id
    next object to be created will have its ids set to XX0, ... X<n-1>
    This is particularly useful for processes that may have several ids
    if they belong to nested namespaces.

The objects covered here are ipc objects and processes.

The patches are still against 2.6.25-rc3-mm1, in the following order:

[PATCH 1/4] adds the procfs facility for next object to be created, this
        object being associated to a single id.
[PATCH 2/4] enhances the procfs facility for objects associated to multiple
        ids (like processes).
[PATCH 3/4] makes use of the specified id (if any) to allocate the new IPC
        object (changes the ipc_addid() path).
[PATCH 4/4] uses the specified id(s) (if any) to set the upid nr(s) for a newly
        allocated process (changes the alloc_pid() path).

Any comment and/or suggestions are welcome.

Regards,
Nadia

--

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

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>

```
---
 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,
+};
+
+
 #ifdef CONFIG_SCHED_DEBUG
 /*
  * Print out various scheduling related per-task fields:
@@ -2391,6 +2462,7 @@ static const struct pid_entry tgid_base_
 #ifdef 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
 #ifdef 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

---

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;
+ }
 }

--
```

_____

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

---

Subject: [RFC][PATCH 3/4] IPC: use the target ID specified in procfs
Posted by Nadia Derbey on Fri, 28 Mar 2008 09:53:12 GMT
View Forum Message <> Reply to Message

[PATCH 03/04]

This patch makes use of the target id specified by a previous write into
/proc/self/next_id as the id to use to allocate the next IPC object.

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

```
---
 include/linux/sysids.h |    7 +++++++
 ipc/util.c             |   39 +++++++++++++++++++++++++++++++++++++++--------
 kernel/nextid.c        |    2 +-
 3 files changed, 39 insertions(+), 9 deletions(-)

Index: linux-2.6.25-rc3-mm1/include/linux/sysids.h
===================================================================
--- linux-2.6.25-rc3-mm1.orig/include/linux/sysids.h 2008-03-28 08:19:49.000000000 +0100
+++ linux-2.6.25-rc3-mm1/include/linux/sysids.h 2008-03-28 08:21:02.000000000 +0100
@@ -37,7 +37,14 @@ struct sys_id {
  long *blocks[0];
 };

+#define next_ipcid(tsk) ((tsk)->next_id     \
+    ? ((tsk)->next_id->nids   \
+     ? ID_AT((tsk)->next_id, 0) \
```

```
+     : -1)    \
+    : -1)
+
 extern ssize_t get_nextid(struct task_struct *, char *);
 extern int set_nextid(struct task_struct *, char *);
+extern void id_blocks_free(struct sys_id *);

 #endif /* _LINUX_SYSIDS_H */
Index: linux-2.6.25-rc3-mm1/kernel/nextid.c
===================================================================
--- linux-2.6.25-rc3-mm1.orig/kernel/nextid.c 2008-03-28 08:20:52.000000000 +0100
+++ linux-2.6.25-rc3-mm1/kernel/nextid.c 2008-03-28 08:21:02.000000000 +0100
@@ -49,7 +49,7 @@ out_undo_partial_alloc:
  return NULL;
 }

-static void id_blocks_free(struct sys_id *ids)
+void id_blocks_free(struct sys_id *ids)
 {
  if (ids == NULL)
   return;
Index: linux-2.6.25-rc3-mm1/ipc/util.c
===================================================================
--- linux-2.6.25-rc3-mm1.orig/ipc/util.c 2008-03-28 08:19:49.000000000 +0100
+++ linux-2.6.25-rc3-mm1/ipc/util.c 2008-03-28 08:21:02.000000000 +0100
@@ -260,6 +260,7 @@ int ipc_get_maxid(struct ipc_ids *ids)
 int ipc_addid(struct ipc_ids* ids, struct kern_ipc_perm* new, int size)
 {
  int id, err;
+ int next_id;

  if (size > IPCMNI)
   size = IPCMNI;
@@ -267,20 +268,42 @@ int ipc_addid(struct ipc_ids* ids, struc
  if (ids->in_use >= size)
   return -ENOSPC;

- err = idr_get_new(&ids->ipcs_idr, new, &id);
- if (err)
-  return err;
+ next_id = next_ipcid(current);
+ if (next_id >= 0) {
+  /* There is a target id specified, try to use it */
+  int new_lid = next_id % SEQ_MULTIPLIER;
+
+  if (next_id !=
+      (new_lid + (next_id / SEQ_MULTIPLIER) * SEQ_MULTIPLIER))
+   return -EINVAL;
```

```
+
+  err = idr_get_new_above(&ids->ipcs_idr, new, new_lid, &id);
+  if (err)
+   return err;
+  if (id != new_lid) {
+   idr_remove(&ids->ipcs_idr, id);
+   return -EBUSY;
+  }
+
+  new->id = next_id;
+  new->seq = next_id / SEQ_MULTIPLIER;
+  id_blocks_free(current->next_id);
+ } else {
+  err = idr_get_new(&ids->ipcs_idr, new, &id);
+  if (err)
+   return err;
+
+  new->seq = ids->seq++;
+  if (ids->seq > ids->seq_max)
+   ids->seq = 0;
+  new->id = ipc_buildid(id, new->seq);
+ }

  ids->in_use++;

  new->cuid = new->uid = current->euid;
  new->gid = new->cgid = current->egid;

- new->seq = ids->seq++;
- if(ids->seq > ids->seq_max)
-  ids->seq = 0;
-
- new->id = ipc_buildid(id, new->seq);
  spin_lock_init(&new->lock);
  new->deleted = 0;
  rcu_read_lock();

--
```

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

Subject: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by Nadia Derbey on Fri, 28 Mar 2008 09:53:13 GMT
View Forum Message <> Reply to Message

[PATCH 04/04]

This patch makes use of the target ids specified by a previous write to
/proc/self/next_id as the ids to use to allocate the next upid nrs.
Upper levels upid nrs that are not specified in next_pids file are left to the
kernel choice.

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

---
 include/linux/pid.h | 2
 kernel/fork.c | 3 -
 kernel/pid.c | 141 ++-------
 3 files changed, 126 insertions(+), 20 deletions(-)

Index: linux-2.6.25-rc3-mm1/include/linux/pid.h
===
--- linux-2.6.25-rc3-mm1.orig/include/linux/pid.h 2008-03-28 08:18:53.000000000 +0100
+++ linux-2.6.25-rc3-mm1/include/linux/pid.h 2008-03-28 08:21:19.000000000 +0100
@@ -119,7 +119,7 @@ extern struct pid *find_get_pid(int nr);
 extern struct pid *find_ge_pid(int nr, struct pid_namespace *);
 int next_pidmap(struct pid_namespace *pid_ns, int last);

-extern struct pid *alloc_pid(struct pid_namespace *ns);
+extern struct pid *alloc_pid(struct pid_namespace *ns, int *retval);
 extern void free_pid(struct pid *pid);

 /*
Index: linux-2.6.25-rc3-mm1/kernel/fork.c
===
--- linux-2.6.25-rc3-mm1.orig/kernel/fork.c 2008-03-28 08:18:53.000000000 +0100
+++ linux-2.6.25-rc3-mm1/kernel/fork.c 2008-03-28 08:21:19.000000000 +0100
@@ -1199,8 +1199,7 @@ static struct task_struct *copy_process(
   goto bad_fork_cleanup_io;

  if (pid != &init_struct_pid) {
- retval = -ENOMEM;
- pid = alloc_pid(task_active_pid_ns(p));
+ pid = alloc_pid(task_active_pid_ns(p), &retval);
   if (!pid)
    goto bad_fork_cleanup_io;

Index: linux-2.6.25-rc3-mm1/kernel/pid.c
===
--- linux-2.6.25-rc3-mm1.orig/kernel/pid.c 2008-03-28 08:18:53.000000000 +0100
+++ linux-2.6.25-rc3-mm1/kernel/pid.c 2008-03-28 08:21:19.000000000 +0100
@@ -122,6 +122,26 @@ static void free_pidmap(struct upid *upi
  atomic_inc(&map->nr_free);

```
 }

+static inline int alloc_pidmap_page(struct pidmap *map)
+{
+ if (unlikely(!map->page)) {
+  void *page = kzalloc(PAGE_SIZE, GFP_KERNEL);
+  /*
+   * Free the page if someone raced with us
+   * installing it:
+   */
+  spin_lock_irq(&pidmap_lock);
+  if (map->page)
+   kfree(page);
+  else
+   map->page = page;
+  spin_unlock_irq(&pidmap_lock);
+  if (unlikely(!map->page))
+   return -1;
+ }
+ return 0;
+}
+
 static int alloc_pidmap(struct pid_namespace *pid_ns)
 {
  int i, offset, max_scan, pid, last = pid_ns->last_pid;
@@ -134,21 +154,8 @@ static int alloc_pidmap(struct pid_names
  map = &pid_ns->pidmap[pid/BITS_PER_PAGE];
  max_scan = (pid_max + BITS_PER_PAGE - 1)/BITS_PER_PAGE - !offset;
  for (i = 0; i <= max_scan; ++i) {
-  if (unlikely(!map->page)) {
-   void *page = kzalloc(PAGE_SIZE, GFP_KERNEL);
-   /*
-    * Free the page if someone raced with us
-    * installing it:
-    */
-   spin_lock_irq(&pidmap_lock);
-   if (map->page)
-    kfree(page);
-   else
-    map->page = page;
-   spin_unlock_irq(&pidmap_lock);
-   if (unlikely(!map->page))
-    break;
-  }
+  if (unlikely(alloc_pidmap_page(map)))
+   break;
   if (likely(atomic_read(&map->nr_free))) {
    do {
```

```
     if (!test_and_set_bit(offset, map->page)) {
@@ -182,6 +189,35 @@ static int alloc_pidmap(struct pid_names
  return -1;
 }

+/*
+ * Return a predefined pid value if successful (ID_AT(pid_l, level)),
+ * -errno else
+ */
+static int alloc_fixed_pidmap(struct pid_namespace *pid_ns,
+    struct sys_id *pid_l, int level)
+{
+ int offset, pid;
+ struct pidmap *map;
+
+ pid = ID_AT(pid_l, level);
+ if (pid < RESERVED_PIDS || pid >= pid_max)
+  return -EINVAL;
+
+ map = &pid_ns->pidmap[pid / BITS_PER_PAGE];
+
+ if (unlikely(alloc_pidmap_page(map)))
+  return -ENOMEM;
+
+ offset = pid & BITS_PER_PAGE_MASK;
+ if (test_and_set_bit(offset, map->page))
+  return -EBUSY;
+
+ atomic_dec(&map->nr_free);
+ pid_ns->last_pid = max(pid_ns->last_pid, pid);
+
+ return pid;
+}
+
 int next_pidmap(struct pid_namespace *pid_ns, int last)
 {
  int offset;
@@ -243,20 +279,91 @@ void free_pid(struct pid *pid)
  call_rcu(&pid->rcu, delayed_put_pid);
 }

-struct pid *alloc_pid(struct pid_namespace *ns)
+/*
+ * Called by alloc_pid() to use a list of predefined ids for the calling
+ * process' upper ns levels.
+ * Returns next pid ns to visit if successful (may be NULL if walked through
+ * the entire pid ns hierarchy).
+ * i is filled with next level to be visited (useful for the error cases).
```

```
+ */
+static struct pid_namespace *set_predefined_pids(struct pid_namespace *ns,
+       struct pid *pid,
+       struct sys_id *pid_l,
+       int *next_level)
+{
+ struct pid_namespace *tmp;
+ int rel_level, i, nr;
+
+ rel_level = pid_l->nids - 1;
+ if (rel_level > ns->level)
+  return ERR_PTR(-EINVAL);
+
+ tmp = ns;
+
+ /*
+  * Use the predefined upid nrs for levels ns->level down to
+  * ns->level - rel_level
+  */
+ for (i = ns->level ; rel_level >= 0; i--, rel_level--) {
+  nr = alloc_fixed_pidmap(tmp, pid_l, rel_level);
+  if (nr < 0) {
+   tmp = ERR_PTR(nr);
+   goto out;
+  }
+
+  pid->numbers[i].nr = nr;
+  pid->numbers[i].ns = tmp;
+  tmp = tmp->parent;
+ }
+
+ id_blocks_free(pid_l);
+
+out:
+ *next_level = i;
+ return tmp;
+}
+
+struct pid *alloc_pid(struct pid_namespace *ns, int *retval)
 {
  struct pid *pid;
  enum pid_type type;
  int i, nr;
  struct pid_namespace *tmp;
  struct upid *upid;
+ struct sys_id *pid_l;

+ *retval = -ENOMEM;
```

```
  pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
  if (!pid)
   goto out;

  tmp = ns;
- for (i = ns->level; i >= 0; i--) {
+ i = ns->level;
+
+ /*
+  * If there is a list of upid nrs specified, use it instead of letting
+  * the kernel chose the upid nrs for us.
+  */
+ pid_l = current->next_id;
+ if (pid_l && pid_l->nids) {
+  /*
+   * returns the next ns to be visited in the following loop
+   * (or NULL if we are done).
+   * i is filled in with the next level to be visited. We need
+   * it to undo things in the error cases.
+   */
+  tmp = set_predefined_pids(ns, pid, pid_l, &i);
+  if (IS_ERR(tmp)) {
+   *retval = PTR_ERR(tmp);
+   goto out_free;
+  }
+ }
+
+ *retval = -ENOMEM;
+ /*
+  * Let the lower levels upid nrs be automatically allocated
+  */
+ for ( ; i >= 0; i--) {
   nr = alloc_pidmap(tmp);
   if (nr < 0)
    goto out_free;

--
```

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

---

Subject: Re: [RFC][PATCH 1/4] Provide a new procfs interface to set next id
Posted by Oren Laadan on Fri, 28 Mar 2008 19:12:51 GMT
View Forum Message <> Reply to Message

On Fri, 28 Mar 2008, Nadia.Derbey@bull.net wrote:

> [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>
>
> ---
> 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,
> +};
```

> +
> +
> #ifdef CONFIG_SCHED_DEBUG
> /*
> * Print out various scheduling related per-task fields:
> @@ -2391,6 +2462,7 @@ static const struct pid_entry tgid_base_
> #ifdef 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
> #ifdef 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);
> +}
```

Since the input format is "LONG <id>", why not have the output format be the same ? This way, if in the future there are other formats (eg CHAR, IPv4 ...). Same comment for the next patch, for LONG<n> format.

(disclaimer: at this point I don't see why one would want to read this file, or usage for other formats...)

```
> +
> +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);

Freeing this memory when the task terminates could be helpful, too.
Probably execve() is another good candidate to free this, just in case.

Oren.

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

---

Subject: Re: [RFC][PATCH 0/4] Object creation with a pre-defined id (v2)
Posted by Oren Laadan on Sat, 29 Mar 2008 00:09:13 GMT
View Forum Message <> Reply to Message

On Fri, 28 Mar 2008, Nadia.Derbey@bull.net wrote:

>
> Hi,
>
> Here is a second version of what has been proposed 2 weeks ago to create
> an object with a pre-defined id (this feature would be used during the
> restart operation) - see thread
https://lists.linux-foundation.org/pipermail/containers/2008-March/thread.html#10287
>
> Main changes since last version:
> . Pavel's suggestion has been integrated; this makes things more readable:
>   alloc_pidmap() is unchanged and a alloc_fixed_pidmap() is added for the
>   predefined ids.
> . Oren's suggestion has been integrated:
>   We now have a single file under /proc/self (/proc/self/next_id).
>   When this file is filled, a structure pointed to by the calling task struct
>   is filled with the id(s).
>   Then, when the object is created, the id(s) present in that structure are
>   used, instead of the default ones.
>   The syntax is one of:
>   . echo "LONG XX" > /proc/self/next_id
>   next object to be created will have an id set to XX
>   . echo "LONG<n> X0 ... X<n-1>" > /proc/self/next_id
>   next object to be created will have its ids set to XX0, ... X<n-1>
>   This is particularly useful for processes that may have several ids
>   if they belong to nested namespaces.

I suggest that there be a way for the process to reset its task->next_id
discarding previous settings, e.g. to recover from an error condition and
prevent subsuquent syscalls from using the task->next_id unintentionally.
Something like "echo RESET > /proc/self/next_id" (or s/RESET/0/ ... etc).

> The objects covered here are ipc objects and processes.
>
> The patches are still against 2.6.25-rc3-mm1, in the following order:
>
> [PATCH 1/4] adds the procfs facility for next object to be created, this
>   object being associated to a single id.
> [PATCH 2/4] enhances the procfs facility for objects associated to multiple
>   ids (like processes).
> [PATCH 3/4] makes use of the specified id (if any) to allocate the new IPC
>   object (changes the ipc_addid() path).
> [PATCH 4/4] uses the specified id(s) (if any) to set the upid nr(s) for a newly
>   allocated process (changes the alloc_pid() path).
>
> Any comment and/or suggestions are welcome.
>
> Regards,

> Nadia
>
> --
>

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

---

Subject: Re: [RFC][PATCH 0/4] Object creation with a pre-defined id (v2)
Posted by Oren Laadan on Sat, 29 Mar 2008 00:13:16 GMT
View Forum Message <> Reply to Message

On Fri, 28 Mar 2008, Nadia.Derbey@bull.net wrote:

>
> Hi,
>
> Here is a second version of what has been proposed 2 weeks ago to create
> an object with a pre-defined id (this feature would be used during the
> restart operation) - see thread
https://lists.linux-foundation.org/pipermail/containers/2008-March/thread.html#10287
>
> Main changes since last version:
> . Pavel's suggestion has been integrated; this makes things more readable:
> alloc_pidmap() is unchanged and a alloc_fixed_pidmap() is added for the
> predefined ids.
> . Oren's suggestion has been integrated:
> We now have a single file under /proc/self (/proc/self/next_id).
> When this file is filled, a structure pointed to by the calling task struct
> is filled with the id(s).
> Then, when the object is created, the id(s) present in that structure are
> used, instead of the default ones.
> The syntax is one of:
> . echo "LONG XX" > /proc/self/next_id
> next object to be created will have an id set to XX
> . echo "LONG<n> X0 ... X<n-1>" > /proc/self/next_id
> next object to be created will have its ids set to XX0, ... X<n-1>
> This is particularly useful for processes that may have several ids
> if they belong to nested namespaces.

I suggest that there be a way for the process to reset its task->next_id
discarding previous settings, e.g. to recover from an error condition and
prevent subsuquent syscalls from using the task->next_id unintentionally.
Something like "echo RESET > /proc/self/next_id" (or s/RESET/0/ ... etc).

Oren.

> The objects covered here are ipc objects and processes.
>
> The patches are still against 2.6.25-rc3-mm1, in the following order:
>
> [PATCH 1/4] adds the procfs facility for next object to be created, this
> object being associated to a single id.
> [PATCH 2/4] enhances the procfs facility for objects associated to multiple
> ids (like processes).
> [PATCH 3/4] makes use of the specified id (if any) to allocate the new IPC
> object (changes the ipc_addid() path).
> [PATCH 4/4] uses the specified id(s) (if any) to set the upid nr(s) for a newly
> allocated process (changes the alloc_pid() path).
>
> Any comment and/or suggestions are welcome.
>
> Regards,
> Nadia
>
> --
>

_____

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

---

Subject: Re: [RFC][PATCH 0/4] Object creation with a pre-defined id (v2)
Posted by serue on Wed, 02 Apr 2008 15:56:53 GMT
View Forum Message <> Reply to Message

Quoting Oren Laadan (orenl@cs.columbia.edu):
>
>
> On Fri, 28 Mar 2008, Nadia.Derbey@bull.net wrote:
>
> >
> > Hi,
> >
> > Here is a second version of what has been proposed 2 weeks ago to create
> > an object with a pre-defined id (this feature would be used during the
> > restart operation) - see thread
https://lists.linux-foundation.org/pipermail/containers/2008-March/thread.html#10287
> >
> > Main changes since last version:
> > . Pavel's suggestion has been integrated; this makes things more readable:
> > alloc_pidmap() is unchanged and a alloc_fixed_pidmap() is added for the
> > predefined ids.

> > . Oren's suggestion has been integrated:
> >   We now have a single file under /proc/self (/proc/self/next_id).
> >   When this file is filled, a structure pointed to by the calling task struct
> >   is filled with the id(s).
> >   Then, when the object is created, the id(s) present in that structure are
> >   used, instead of the default ones.
> >   The syntax is one of:
> >   . echo "LONG XX" > /proc/self/next_id
> >   next object to be created will have an id set to XX
> >   . echo "LONG<n> X0 ... X<n-1>" > /proc/self/next_id
> >   next object to be created will have its ids set to XX0, ... X<n-1>
> >   This is particularly useful for processes that may have several ids
> >   if they belong to nested namespaces.
>
> I suggest that there be a way for the process to reset its task->next_id
> discarding previous settings, e.g. to recover from an error condition and
> prevent subsuquent syscalls from using the task->next_id unintentionally.
> Something like "echo RESET > /proc/self/next_id" (or s/RESET/0/ ... etc).

Makes sense, and should be trivial enough.

If there are no objections to this approach on this list, should you be
sending this patchset (addressing Oren's comments) on to lkml? Or is it
too soon for that?

thanks,
-serge

> > The objects covered here are ipc objects and processes.
> >
> > The patches are still against 2.6.25-rc3-mm1, in the following order:
> >
> > [PATCH 1/4] adds the procfs facility for next object to be created, this
> >   object being associated to a single id.
> > [PATCH 2/4] enhances the procfs facility for objects associated to multiple
> >   ids (like processes).
> > [PATCH 3/4] makes use of the specified id (if any) to allocate the new IPC
> >   object (changes the ipc_addid() path).
> > [PATCH 4/4] uses the specified id(s) (if any) to set the upid nr(s) for a newly
> >   allocated process (changes the alloc_pid() path).
> >
> > Any comment and/or suggestions are welcome.
> >
> > Regards,
> > Nadia
> >
> > --
> >

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

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