Subject: [RFC][PATCH 0/4] Object creation with a specified id
Posted by Nadia Derbey on Fri, 04 Apr 2008 14:51:29 GMT
View Forum Message <> Reply to Message

Hi,

When restarting a process that has been previously checkpointed, that process should keep on using some of its ids (such as its process id, or sysV ipc ids).

This patch provides a feature that can help ensuring this saved state reuse: it makes it possible to create an object with a pre-defined id.

A first implementation had been proposed 2 months ago. It consisted in changing an object's id after it had been created.

Here is a second implementation based on Oren Ladaan's idea: Oren's suggestion was to force an object's id during its creation, rather than 1. create it, 2. change its id.

A new file is created in procfs: /proc/self/next_id.

When this file is filled with and id value, a structure pointed to by the calling task struct is filled with that id.

Then, when an object supporting this feature is created, the id present in that new structure is used, instead of the default one.

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.
Today, the ids are specified as long, but having a type string specified in the next_id file makes it possible to cover more types in the future, if needed.

The patches are 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()/alloc_pidmap() paths).

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 2/4] Provide a new procfs interface to set next upid nr(s)
Posted by Nadia Derbey on Fri, 04 Apr 2008 14:51:31 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_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>

---
include/linux/sysids.h | 27 ++++++++
kernel/nextid.c | 150 +++-------

```
 2 files changed, 155 insertions(+), 22 deletions(-)

Index: linux-2.6.25-rc8-mm1/include/linux/sysids.h
===================================================================
--- linux-2.6.25-rc8-mm1.orig/include/linux/sysids.h 2008-04-04 13:53:04.000000000 +0200
+++ linux-2.6.25-rc8-mm1/include/linux/sysids.h 2008-04-04 14:18:04.000000000 +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 <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 *, size_t);
Index: linux-2.6.25-rc8-mm1/kernel/nextid.c
===================================================================
--- linux-2.6.25-rc8-mm1.orig/kernel/nextid.c 2008-04-04 13:59:59.000000000 +0200
+++ linux-2.6.25-rc8-mm1/kernel/nextid.c 2008-04-04 14:28:13.000000000 +0200
@@ -13,38 +13,138 @@
```

```
+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 count = 0;
  struct sys_id *sid;
+ char *bufptr = buffer;
+ int i;

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

- return snprintf(buffer, size, "LONG %ld\n", sid->id);
+ count = sprintf(bufptr, "LONGS (%d) ", sid->nids);
+
+ for (i = 0; i < sid->nids - 1; i++)
+  count += sprintf(&bufptr[count], "%ld ", ID_AT(sid, i));
+
+ count += sprintf(&bufptr[count], "%ld\n", 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++;
  }

- 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 +155,8 @@ 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 +165,14 @@ int reset_nextid(struct task_struct *tas

 /*
  * Parses a line written to /proc/self/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 +180,15 @@ 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
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, 04 Apr 2008 14:51:32 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             |   40 ++++++++++++++++++++++++++++++++--------
 kernel/nextid.c        |    2 +-
 3 files changed, 40 insertions(+), 9 deletions(-)

Index: linux-2.6.25-rc8-mm1/include/linux/sysids.h
===================================================================
--- linux-2.6.25-rc8-mm1.orig/include/linux/sysids.h 2008-04-04 14:18:04.000000000 +0200
+++ linux-2.6.25-rc8-mm1/include/linux/sysids.h 2008-04-04 14:37:45.000000000 +0200
@@ -37,9 +37,16 @@ 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 *, size_t);
 extern int set_nextid(struct task_struct *, char *);
 extern int reset_nextid(struct task_struct *);
+extern void id_blocks_free(struct sys_id *);
```

```
 static inline void exit_nextid(struct task_struct *tsk)
 {
Index: linux-2.6.25-rc8-mm1/kernel/nextid.c
===================================================================
--- linux-2.6.25-rc8-mm1.orig/kernel/nextid.c 2008-04-04 14:28:13.000000000 +0200
+++ linux-2.6.25-rc8-mm1/kernel/nextid.c 2008-04-04 14:38:38.000000000 +0200
@@ -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-rc8-mm1/ipc/util.c
===================================================================
--- linux-2.6.25-rc8-mm1.orig/ipc/util.c 2008-04-04 13:11:37.000000000 +0200
+++ linux-2.6.25-rc8-mm1/ipc/util.c 2008-04-04 14:41:53.000000000 +0200
@@ -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 +268,43 @@ 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);
+  current->next_id = NULL;
+ } 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();

--
```

Subject: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by Nadia Derbey on Fri, 04 Apr 2008 14:51:33 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-rc8-mm1/include/linux/pid.h
===
--- linux-2.6.25-rc8-mm1.orig/include/linux/pid.h 2008-04-04 13:11:37.000000000 +0200
+++ linux-2.6.25-rc8-mm1/include/linux/pid.h 2008-04-04 14:54:09.000000000 +0200
@@ -121,7 +121,7 @@ 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-rc8-mm1/kernel/fork.c
===
--- linux-2.6.25-rc8-mm1.orig/kernel/fork.c 2008-04-04 14:00:35.000000000 +0200
+++ linux-2.6.25-rc8-mm1/kernel/fork.c 2008-04-04 14:54:43.000000000 +0200
@@ -1200,8 +1200,7 @@ 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-rc8-mm1/kernel/pid.c
===
--- linux-2.6.25-rc8-mm1.orig/kernel/pid.c 2008-04-04 13:11:39.000000000 +0200
+++ linux-2.6.25-rc8-mm1/kernel/pid.c 2008-04-04 14:59:24.000000000 +0200
@@ -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;
+  }
+  current->next_id = NULL;
+ }
+
+ *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;

--
```

Subject: Re: [RFC][PATCH 0/4] Object creation with a specified id
Posted by Nick Andrew on Tue, 15 Apr 2008 03:06:23 GMT
View Forum Message <> Reply to Message

On Fri, Apr 04, 2008 at 04:51:29PM +0200, Nadia.Derbey@bull.net wrote:
> . 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.

How do you handle race conditions, i.e. you specify the ID for the
next object to be created, and then some other thread goes and creates
an object before your thread creates one?

Nick.
--
PGP Key ID = 0x418487E7 http://www.nick-andrew.net/
PGP Key fingerprint = B3ED 6894 8E49 1770 C24A 67E3 6266 6EB9 4184 87E7

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