Subject: [RFC][PATCH 0/4] Object creation with a specified id
Posted by Nadia Derbey on Mon, 10 Mar 2008 13:50:54 GMT
View Forum Message <> Reply to Message

A couple of weeks ago, a discussion has started after Pierre's proposal for
a new syscall to change an ipc id (see thread
http://lkml.org/lkml/2008/1/29/209).

Oren's suggestion was to force an object's id during its creation, rather
than 1. create it, 2. change its id.

So here is an implementation of what Oren has suggested.

2 new files are defined under /proc/self:
  . next_ipcid --> next id to use for ipc object creation
  . next_pids --> next upid nr(s) to use for next task to be forked
                (see patch #2 for more details).

When one of these files (or both of them) is filled, a structure pointed to
by the calling task struct is filled with these ids.

Then, when the object is created, the id(s) present in that structure are
used, instead of the default ones.

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

[PATCH 1/4] adds the procfs facility for next ipc to be created.
[PATCH 2/4] adds the procfs facility for next task to be forked.
[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.

Cc-ing Pavel and Sukadev, since they are the pid namespace authors.

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 ipc id
Posted by Nadia Derbey on Mon, 10 Mar 2008 13:50:55 GMT
View Forum Message <> Reply to Message

[PATCH 01/04]

This patch proposes the procfs facilities needed to feed the id for the
next ipc object to be allocated.

if an
echo XX > /proc/self/next_ipcid
is issued, next ipc object to be created will have XX as its id.

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

```
---
 fs/proc/base.c          |   66 +++++++++++++++++++++++++++++++++++++++++++++++++
 include/linux/sched.h   |    3 ++
 include/linux/sysids.h  |   20 ++++++++++++++
 kernel/Makefile         |    2 -
 kernel/set_nextid.c     |   31 +++++++++++++++++++++++
 5 files changed, 121 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 2008-03-10 11:39:10.000000000 +0100
@@ -0,0 +1,20 @@
+/*
+ * include/linux/sysids.h
+ *
+ * Definitions to support object creation with predefined id.
+ *
+ */
+
+#ifndef _LINUX_SYSIDS_H
+#define _LINUX_SYSIDS_H
+
+#define SYS_ID_IPC 1
+
+struct sys_id {
+ int flag; /* which id should be set */
+ int ipc;
+};
+
+extern int ipc_set_nextid(struct task_struct *, int id);
+
+#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 2008-03-10 11:22:20.000000000 +0100
@@ -76,6 +76,7 @@
 #include <linux/oom.h>
 #include <linux/elf.h>
 #include <linux/pid_namespace.h>
+#include <linux/ctype.h>
 #include "internal.h"

 /* NOTE:
@@ -1080,6 +1081,69 @@ static const struct file_operations proc
 #endif


+static ssize_t next_ipcid_read(struct file *file, char __user *buf,
+    size_t count, loff_t *ppos)
+{
+ struct task_struct *task;
+ char buffer[PROC_NUMBUF];
+ size_t len;
+ struct sys_id *sid;
+ int next_ipcid;
+
+ task = get_proc_task(file->f_path.dentry->d_inode);
+ if (!task)
+  return -ESRCH;
```

```
+
+ sid = task->next_id;
+ next_ipcid = (sid) ? ((sid->flag & SYS_ID_IPC) ? sid->ipc : -1)
+   : -1;
+
+ put_task_struct(task);
+
+ len = snprintf(buffer, sizeof(buffer), "%i\n", next_ipcid);
+
+ return simple_read_from_buffer(buf, count, ppos, buffer, len);
+}
+
+static ssize_t next_ipcid_write(struct file *file, const char __user *buf,
+    size_t count, loff_t *ppos)
+{
+ struct inode *inode = file->f_path.dentry->d_inode;
+ char buffer[PROC_NUMBUF], *end;
+ int next_ipcid;
+ int rc;
+
+ if (pid_task(proc_pid(inode), PIDTYPE_PID) != current)
+  return -EPERM;
+
+ memset(buffer, 0, sizeof(buffer));
+ if (count > sizeof(buffer) - 1)
+  count = sizeof(buffer) - 1;
+ if (copy_from_user(buffer, buf, count))
+  return -EFAULT;
+
+ next_ipcid = simple_strtol(buffer, &end, 0);
+ if (next_ipcid < 0 || end == buffer)
+  return -EINVAL;
+
+ while (isspace(*end))
+  end++;
+
+ rc = ipc_set_nextid(current, next_ipcid);
+ if (rc)
+  return rc;
+
+ if (end - buffer == 0)
+  return -EIO;
+ return end - buffer;
+}
+
+static const struct file_operations proc_next_ipcid_operations = {
+ .read  = next_ipcid_read,
+ .write  = next_ipcid_write,
```

```
+};
+
+
 #ifdef CONFIG_SCHED_DEBUG
 /*
  * Print out various scheduling related per-task fields:
@@ -2391,6 +2455,7 @@ static const struct pid_entry tgid_base_
 #ifdef CONFIG_TASK_IO_ACCOUNTING
  INF("io", S_IRUGO, pid_io_accounting),
 #endif
+ REG("next_ipcid", S_IRUGO|S_IWUSR, next_ipcid),
 };

 static int proc_tgid_base_readdir(struct file * filp,
@@ -2716,6 +2781,7 @@ 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_ipcid", S_IRUGO|S_IWUSR, next_ipcid),
 };

 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 2008-03-10 09:41:27.000000000 +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 set_nextid.o

 obj-$(CONFIG_SYSCTL_SYSCALL_CHECK) += sysctl_check.o
 obj-$(CONFIG_STACKTRACE) += stacktrace.o
Index: linux-2.6.25-rc3-mm1/kernel/set_nextid.c
===================================================================
--- /dev/null 1970-01-01 00:00:00.000000000 +0000
+++ linux-2.6.25-rc3-mm1/kernel/set_nextid.c 2008-03-10 10:09:47.000000000 +0100
@@ -0,0 +1,31 @@
+/*
+ * linux/kernel/set_nextid.c
+ *
+ *
+ * Provide the XXX_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>
+
+
+
+int ipc_set_nextid(struct task_struct *task, int id)
+{
+ struct sys_id *sid;
+
+ sid = task->next_id;
+ if (!sid) {
+  sid = kzalloc(sizeof(*sid), GFP_KERNEL);
+  if (!sid)
+   return -ENOMEM;
+  task->next_id = sid;
+ }
+
+ sid->ipc = id;
+ sid->flag |= SYS_ID_IPC;
+
+ return 0;
+}
+

--
```

_____

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 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;
+}

--
```

_____
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 Mon, 10 Mar 2008 13:50:57 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_ipcid as the id to use to allocate the next IPC object.

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

```
---
 ipc/util.c |   39 ++++++++++++++++++++++++++++++++-------
 1 file changed, 31 insertions(+), 8 deletions(-)

Index: linux-2.6.25-rc3-mm1/ipc/util.c
===================================================================
--- linux-2.6.25-rc3-mm1.orig/ipc/util.c 2008-03-10 13:10:33.000000000 +0100
+++ linux-2.6.25-rc3-mm1/ipc/util.c 2008-03-10 13:11:24.000000000 +0100
@@ -35,6 +35,7 @@
 #include <linux/rwsem.h>
 #include <linux/memory.h>
 #include <linux/ipc_namespace.h>
+#include <linux/sysids.h>

 #include <asm/unistd.h>

@@ -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;
+ /* If there is a target id specified, try to use it */
+ if (current->next_id && (current->next_id->flag & SYS_ID_IPC)) {
+  int new_id = current->next_id->ipc_id;
+  int new_lid = new_id % SEQ_MULTIPLIER;
+
+  if (new_id !=
+      (new_lid + (new_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 = new_id;
+  new->seq = new_id / SEQ_MULTIPLIER;
+  current->next_id->flag &= ~SYS_ID_IPC;
+ } 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 Mon, 10 Mar 2008 13:50:58 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_pids 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>

---
 kernel/fork.c | 5 ++-
 kernel/pid.c | 80 ++--------
 2 files changed, 73 insertions(+), 12 deletions(-)

Index: linux-2.6.25-rc3-mm1/kernel/pid.c
===
--- linux-2.6.25-rc3-mm1.orig/kernel/pid.c 2008-03-10 09:19:02.000000000 +0100
+++ linux-2.6.25-rc3-mm1/kernel/pid.c 2008-03-10 13:45:52.000000000 +0100
@@ -122,14 +122,26 @@ static void free_pidmap(struct upid *upi
  atomic_inc(&map->nr_free);
 }

-static int alloc_pidmap(struct pid_namespace *pid_ns)
+static int alloc_pidmap(struct pid_namespace *pid_ns, struct pid_list *pid_l,
+   int level)
 {
  int i, offset, max_scan, pid, last = pid_ns->last_pid;
  struct pidmap *map;

- pid = last + 1;
- if (pid >= pid_max)
- pid = RESERVED_PIDS;
+ if (!pid_l) {
+ pid = last + 1;
+ if (pid >= pid_max)
+   pid = RESERVED_PIDS;
+ } else {
+ /*

```
+   * There's a target pid, so use it instead
+   */
+  BUG_ON(level < 0);
+  pid = PID_AT(pid_l, level);
+  if (pid >= pid_max)
+   return -EINVAL;
+ }
+
  offset = pid & BITS_PER_PAGE_MASK;
  map = &pid_ns->pidmap[pid/BITS_PER_PAGE];
  max_scan = (pid_max + BITS_PER_PAGE - 1)/BITS_PER_PAGE - !offset;
@@ -153,9 +165,16 @@ static int alloc_pidmap(struct pid_names
    do {
     if (!test_and_set_bit(offset, map->page)) {
      atomic_dec(&map->nr_free);
-     pid_ns->last_pid = pid;
+     if (!pid_l)
+      pid_ns->last_pid = pid;
+     else
+      pid_ns->last_pid = max(last,
+         pid);
      return pid;
     }
+    if (pid_l)
+     /* Target pid is already in use */
+     return -EBUSY;
     offset = find_next_offset(map, offset);
     pid = mk_pid(pid_ns, map, offset);
    /*
@@ -179,7 +198,7 @@ static int alloc_pidmap(struct pid_names
   }
   pid = mk_pid(pid_ns, map, offset);
  }
- return -1;
+ return -ENOMEM;
 }

 int next_pidmap(struct pid_namespace *pid_ns, int last)
@@ -250,14 +269,55 @@ struct pid *alloc_pid(struct pid_namespa
  int i, nr;
  struct pid_namespace *tmp;
  struct upid *upid;
+ struct pid_list *pid_l = NULL;
+ int rel_level = -1;
+
+ /*
+  * If there is a list of upid nrs specified, use it instead of letting
+  * the kernel chose them for us.
```

```
+  */
+ if (current->next_id && (current->next_id->flag & SYS_ID_PID)) {
+  pid_l = current->next_id->pid_ids;
+  BUG_ON(!pid_l);
+  rel_level = pid_l->npids - 1;
+  if (rel_level > ns->level) {
+   pid = ERR_PTR(-EINVAL);
+   goto out;
+  }
+ }

  pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
- if (!pid)
+ if (!pid) {
+  pid = ERR_PTR(-ENOMEM);
   goto out;
+ }

  tmp = ns;
- for (i = ns->level; i >= 0; i--) {
-  nr = alloc_pidmap(tmp);
+ /*
+  * 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_pidmap(tmp, pid_l, rel_level);
+  if (nr < 0)
+   goto out_free;
+
+  pid->numbers[i].nr = nr;
+  pid->numbers[i].ns = tmp;
+  tmp = tmp->parent;
+ }
+
+ if (pid_l) {
+  current->next_id->flag &= ~SYS_ID_PID;
+  pids_free(pid_l);
+  current->next_id->pid_ids = NULL;
+ }
+
+ /*
+  * Let the lower levels upid nrs be automatically allocated
+  */
+ for ( ; i >= 0; i--) {
+  nr = alloc_pidmap(tmp, NULL, -1);
   if (nr < 0)
    goto out_free;
```

```
@@ -288,7 +348,7 @@ out_free:
   free_pidmap(pid->numbers + i);

  kmem_cache_free(ns->pid_cachep, pid);
- pid = NULL;
+ pid = ERR_PTR(nr);
  goto out;
 }

Index: linux-2.6.25-rc3-mm1/kernel/fork.c
===================================================================
--- linux-2.6.25-rc3-mm1.orig/kernel/fork.c 2008-03-10 09:19:01.000000000 +0100
+++ linux-2.6.25-rc3-mm1/kernel/fork.c 2008-03-10 13:48:03.000000000 +0100
@@ -1197,10 +1197,11 @@ 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));
-  if (!pid)
+  if (IS_ERR(pid)) {
+   retval = PTR_ERR(pid);
    goto bad_fork_cleanup_io;
+  }

   if (clone_flags & CLONE_NEWPID) {
    retval = pid_ns_prepare_proc(task_active_pid_ns(p));

--
```

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

---

Subject: Re: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by Pavel Emelianov on Tue, 11 Mar 2008 12:04:39 GMT
View Forum Message <> Reply to Message

Nadia.Derbey@bull.net wrote:
> @@ -122,14 +122,26 @@ static void free_pidmap(struct upid *upi
>   atomic_inc(&map->nr_free);
> }
>
> -static int alloc_pidmap(struct pid_namespace *pid_ns)
> +static int alloc_pidmap(struct pid_namespace *pid_ns, struct pid_list *pid_l,
> +   int level)

```
>  {
>   int i, offset, max_scan, pid, last = pid_ns->last_pid;
>   struct pidmap *map;
>
> - pid = last + 1;
> - if (pid >= pid_max)
> -  pid = RESERVED_PIDS;
> + if (!pid_l) {
> +  pid = last + 1;
> +  if (pid >= pid_max)
> +   pid = RESERVED_PIDS;
> + } else {
> +  /*
> +   * There's a target pid, so use it instead
> +   */
> +  BUG_ON(level < 0);
> +  pid = PID_AT(pid_l, level);
> +  if (pid >= pid_max)
> +   return -EINVAL;
> + }
> +
>   offset = pid & BITS_PER_PAGE_MASK;
>   map = &pid_ns->pidmap[pid/BITS_PER_PAGE];
>   max_scan = (pid_max + BITS_PER_PAGE - 1)/BITS_PER_PAGE - !offset;
> @@ -153,9 +165,16 @@ static int alloc_pidmap(struct pid_names
>     do {
>      if (!test_and_set_bit(offset, map->page)) {
>       atomic_dec(&map->nr_free);
> -     pid_ns->last_pid = pid;
> +     if (!pid_l)
> +      pid_ns->last_pid = pid;
> +     else
> +      pid_ns->last_pid = max(last,
> +         pid);
>       return pid;
>      }
> +    if (pid_l)
> +     /* Target pid is already in use */
> +     return -EBUSY;
>      offset = find_next_offset(map, offset);
>      pid = mk_pid(pid_ns, map, offset);
>     /*
> @@ -179,7 +198,7 @@ static int alloc_pidmap(struct pid_names
>    }
>    pid = mk_pid(pid_ns, map, offset);
>   }
> - return -1;
> + return -ENOMEM;
```

> }
>
> int next_pidmap(struct pid_namespace *pid_ns, int last)

As fas as this particular piece of code is concerned this all can
be shrunk down to

```
static int set_vpidmap(struct pid_namespace *ns, int pid)
{
	int offset;
	pidmap_t *map;

	offset = pid & BITS_PER_PAGE_MASK;
	map = ns->pidmap + vpid / BITS_PER_PAGE;

	if (unlikely(alloc_pidmap_page(map)))
		return -ENOMEM;

	if (test_and_set_bit(offset, map->page))
		return -EEXIST;

	atomic_dec(&map->nr_free);
	return pid;
}
```

where the alloc_pidmap_page is a consolidated part of code from alloc_pidmap.

And I'm scared of what the alloc_pid is going to become.

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

---

Subject: Re: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by Nadia Derbey on Tue, 11 Mar 2008 15:28:45 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> Nadia.Derbey@bull.net wrote:
>
>>@@ -122,14 +122,26 @@ static void free_pidmap(struct upid *upi
>> atomic_inc(&map->nr_free);
>> }
>>
>>-static int alloc_pidmap(struct pid_namespace *pid_ns)
>>+static int alloc_pidmap(struct pid_namespace *pid_ns, struct pid_list *pid_l,
>>+   int level)

```
>> {
>>  int i, offset, max_scan, pid, last = pid_ns->last_pid;
>>  struct pidmap *map;
>>
>>- pid = last + 1;
>>- if (pid >= pid_max)
>>-  pid = RESERVED_PIDS;
>>+ if (!pid_l) {
>>+  pid = last + 1;
>>+  if (pid >= pid_max)
>>+   pid = RESERVED_PIDS;
>>+ } else {
>>+  /*
>>+   * There's a target pid, so use it instead
>>+   */
>>+  BUG_ON(level < 0);
>>+  pid = PID_AT(pid_l, level);
>>+  if (pid >= pid_max)
>>+   return -EINVAL;
>>+ }
>>+
>>  offset = pid & BITS_PER_PAGE_MASK;
>>  map = &pid_ns->pidmap[pid/BITS_PER_PAGE];
>>  max_scan = (pid_max + BITS_PER_PAGE - 1)/BITS_PER_PAGE - !offset;
>>@@ -153,9 +165,16 @@ static int alloc_pidmap(struct pid_names
>>    do {
>>     if (!test_and_set_bit(offset, map->page)) {
>>      atomic_dec(&map->nr_free);
>>-     pid_ns->last_pid = pid;
>>+     if (!pid_l)
>>+      pid_ns->last_pid = pid;
>>+     else
>>+      pid_ns->last_pid = max(last,
>>+         pid);
>>      return pid;
>>     }
>>+    if (pid_l)
>>+     /* Target pid is already in use */
>>+     return -EBUSY;
>>     offset = find_next_offset(map, offset);
>>     pid = mk_pid(pid_ns, map, offset);
>>    /*
>>@@ -179,7 +198,7 @@ static int alloc_pidmap(struct pid_names
>>   }
>>   pid = mk_pid(pid_ns, map, offset);
>>  }
>>- return -1;
>>+ return -ENOMEM;
```

>> }
>>
>> int next_pidmap(struct pid_namespace *pid_ns, int last)
>
>
> As fas as this particular piece of code is concerned this all can
> be shrunk down to
>
> static int set_vpidmap(struct pid_namespace *ns, int pid)
> {
> int offset;
> pidmap_t *map;
>
> offset = pid & BITS_PER_PAGE_MASK;
> map = ns->pidmap + vpid / BITS_PER_PAGE;
>
> if (unlikely(alloc_pidmap_page(map)))
> return -ENOMEM;
>
> if (test_and_set_bit(offset, map->page))
> return -EEXIST;
>
> atomic_dec(&map->nr_free);
> return pid;
> }
>
> where the alloc_pidmap_page is a consolidated part of code from alloc_pidmap.
>
> And I'm scared of what the alloc_pid is going to become.
>
>

It's true that I made alloc_pid() become uggly, but this patchset was
more intended to continue a discussion.

What we could do is the following (not compiled, not tested...):

```
struct pid *alloc_pid(struct pid_namespace *ns)
{
        struct pid *pid;
        enum pid_type type;
        int i, nr;
        struct pid_namespace *tmp;
        struct upid *upid;

        pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
        if (!pid) {
                pid = ERR_PTR(-ENOMEM);
```

```
                goto out;
        }

        tmp = ns;
        i = ns->level;

        if (current->next_id && (current->next_id->flag & SYS_ID_PID)) {
                tmp = set_predefined_pids(ns,
                                current->next_id->pid_ids);
                if (IS_ERR(tmp)) {
                        nr = PTR_ERR(tmp);
                        goto out_free;
                }
        }

        /*
         * Let the lower levels upid nrs be automatically allocated
         */
        for ( ; i >= 0; i--) {
                nr = alloc_pidmap(tmp, NULL, -1);
                if (nr < 0)
                        goto out_free;
....
```

which would only add a test and a function call to alloc_pid() ==> more readable.
with set_predefined_pids defined as follows (still not compiled, not tested, ...):

```
struct pid_namespace *set_predefined_pids(struct pid_namespace *ns,
                                struct pid_list *pid_l)
{
        int rel_level;

        BUG_ON(!pid_l);

        rel_level = pid_l->npids - 1;
        if (rel_level > ns->level)
                return ERR_PTR(-EINVAL);

        /*
         * Use the predefined upid nrs for levels ns->level down to
         * ns->level - rel_level
         */
        for ( ; rel_level >= 0; i--, rel_level--) {
                nr = alloc_pidmap(tmp, pid_l, rel_level);
                if (nr < 0)
                        return ERR_PTR(nr);
```

```
                pid->numbers[i].nr = nr;
                pid->numbers[i].ns = tmp;
                tmp = tmp->parent;
        }

        current->next_id->flag &= ~SYS_ID_PID;
        pids_free(pid_l);
        current->next_id->pid_ids = NULL;

        return tmp;
}
```

Don't you think that mixing this with your 1st proposal (the
set_vpidmap() one), would make things look better?

Regards,
Nadia

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

---

Subject: Re: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by Pavel Emelianov on Tue, 11 Mar 2008 15:37:37 GMT
View Forum Message <> Reply to Message

Nadia Derbey wrote:
> Pavel Emelyanov wrote:
>> Nadia.Derbey@bull.net wrote:
>>
>>> @@ -122,14 +122,26 @@ static void free_pidmap(struct upid *upi
>>> atomic_inc(&map->nr_free);
>>> }
>>>
>>> -static int alloc_pidmap(struct pid_namespace *pid_ns)
>>> +static int alloc_pidmap(struct pid_namespace *pid_ns, struct pid_list *pid_l,
>>> +   int level)
>>> {
>>> int i, offset, max_scan, pid, last = pid_ns->last_pid;
>>> struct pidmap *map;
>>>
>>> - pid = last + 1;
>>> - if (pid >= pid_max)
>>> - pid = RESERVED_PIDS;
>>> + if (!pid_l) {

```
>>> +  pid = last + 1;
>>> +  if (pid >= pid_max)
>>> +   pid = RESERVED_PIDS;
>>> + } else {
>>> +  /*
>>> +   * There's a target pid, so use it instead
>>> +   */
>>> +  BUG_ON(level < 0);
>>> +  pid = PID_AT(pid_l, level);
>>> +  if (pid >= pid_max)
>>> +   return -EINVAL;
>>> + }
>>> +
>>>  offset = pid & BITS_PER_PAGE_MASK;
>>>  map = &pid_ns->pidmap[pid/BITS_PER_PAGE];
>>>  max_scan = (pid_max + BITS_PER_PAGE - 1)/BITS_PER_PAGE - !offset;
>>> @@ -153,9 +165,16 @@ static int alloc_pidmap(struct pid_names
>>>    do {
>>>     if (!test_and_set_bit(offset, map->page)) {
>>>      atomic_dec(&map->nr_free);
>>> -     pid_ns->last_pid = pid;
>>> +     if (!pid_l)
>>> +      pid_ns->last_pid = pid;
>>> +     else
>>> +      pid_ns->last_pid = max(last,
>>> +         pid);
>>>      return pid;
>>>     }
>>> +   if (pid_l)
>>> +    /* Target pid is already in use */
>>> +    return -EBUSY;
>>>     offset = find_next_offset(map, offset);
>>>     pid = mk_pid(pid_ns, map, offset);
>>>    /*
>>> @@ -179,7 +198,7 @@ static int alloc_pidmap(struct pid_names
>>>   }
>>>   pid = mk_pid(pid_ns, map, offset);
>>>  }
>>> - return -1;
>>> + return -ENOMEM;
>>> }
>>>
>>> int next_pidmap(struct pid_namespace *pid_ns, int last)
>>
>> As fas as this particular piece of code is concerned this all can
>> be shrunk down to
>>
>> static int set_vpidmap(struct pid_namespace *ns, int pid)
```

```
>> {
>>         int offset;
>>         pidmap_t *map;
>>
>>         offset = pid & BITS_PER_PAGE_MASK;
>>         map = ns->pidmap + vpid / BITS_PER_PAGE;
>>
>>         if (unlikely(alloc_pidmap_page(map)))
>>                 return -ENOMEM;
>>
>>         if (test_and_set_bit(offset, map->page))
>>                 return -EEXIST;
>>
>>         atomic_dec(&map->nr_free);
>>         return pid;
>> }
>>
>> where the alloc_pidmap_page is a consolidated part of code from alloc_pidmap.
>>
>> And I'm scared of what the alloc_pid is going to become.
>>
>>
>
> It's true that I made alloc_pid() become uggly, but this patchset was
> more intended to continue a discussion.
>
> What we could do is the following (not compiled, not tested...):
>
> struct pid *alloc_pid(struct pid_namespace *ns)
> {
>         struct pid *pid;
>         enum pid_type type;
>         int i, nr;
>         struct pid_namespace *tmp;
>         struct upid *upid;
>
>         pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
>         if (!pid) {
>                 pid = ERR_PTR(-ENOMEM);
>                 goto out;
>         }
>
>         tmp = ns;
>         i = ns->level;
>
>         if (current->next_id && (current->next_id->flag & SYS_ID_PID)) {
>                 tmp = set_predefined_pids(ns,
>                                 current->next_id->pid_ids);
```

```
>                 if (IS_ERR(tmp)) {
>                         nr = PTR_ERR(tmp);
>                         goto out_free;
>                 }
>         }
>
>         /*
>          * Let the lower levels upid nrs be automatically allocated
>          */
>         for ( ; i >= 0; i--) {
>                 nr = alloc_pidmap(tmp, NULL, -1);
>                 if (nr < 0)
>                         goto out_free;
> ....
>
> which would only add a test and a function call to alloc_pid() ==> more
> readable.
> with set_predefined_pids defined as follows (still not compiled, not
> tested, ...):
>
> struct pid_namespace *set_predefined_pids(struct pid_namespace *ns,
>                                           struct pid_list *pid_l)
> {
>         int rel_level;
>
>         BUG_ON(!pid_l);
>
>         rel_level = pid_l->npids - 1;
>         if (rel_level > ns->level)
>                 return ERR_PTR(-EINVAL);
>
>         /*
>          * Use the predefined upid nrs for levels ns->level down to
>          * ns->level - rel_level
>          */
>         for ( ; rel_level >= 0; i--, rel_level--) {
>                 nr = alloc_pidmap(tmp, pid_l, rel_level);
>                 if (nr < 0)
>                         return ERR_PTR(nr);
>
>                 pid->numbers[i].nr = nr;
>                 pid->numbers[i].ns = tmp;
>                 tmp = tmp->parent;
>         }
>
>         current->next_id->flag &= ~SYS_ID_PID;
>         pids_free(pid_l);
>         current->next_id->pid_ids = NULL;
```

>
>   return tmp;
> }
>
>
> Don't you think that mixing this with your 1st proposal (the
> set_vpidmap() one), would make things look better?

I'd prefer seeing

```
--- a/kernel/pid.c
+++ b/kernel/pid.c
@@ -247,7 +247,7 @@ struct pid *alloc_pid(struct pid_namespace *ns)
 {
        struct pid *pid;
        enum pid_type type;
-       int i, nr;
+       int i, nr, req_nr;
        struct pid_namespace *tmp;
        struct upid *upid;

@@ -257,7 +257,11 @@ struct pid *alloc_pid(struct pid_namespace *ns)

        tmp = ns;
        for (i = ns->level; i >= 0; i--) {
-               nr = alloc_pidmap(tmp);
+               req_nr = get_required_pidnr(ns, i);
+               if (req_nr > 0)
+                       nr = set_pidmap(tmp, req_nr);
+               else
+                       nr = alloc_pidmap(tmp);
                if (nr < 0)
                        goto out_free;
```

in alloc_pid() and not much than that.

> Regards,
> Nadia
>

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

Subject: Re: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by Nadia Derbey on Tue, 11 Mar 2008 15:55:04 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> Nadia Derbey wrote:
>
>>Pavel Emelyanov wrote:
>>
>>>Nadia.Derbey@bull.net wrote:
>>>
>>>
>>>>@@ -122,14 +122,26 @@ static void free_pidmap(struct upid *upi
>>>> atomic_inc(&map->nr_free);
>>>>}
>>>>
>>>>-static int alloc_pidmap(struct pid_namespace *pid_ns)
>>>>+static int alloc_pidmap(struct pid_namespace *pid_ns, struct pid_list *pid_l,
>>>>+ int level)
>>>>{
>>>> int i, offset, max_scan, pid, last = pid_ns->last_pid;
>>>> struct pidmap *map;
>>>>
>>>>- pid = last + 1;
>>>>- if (pid >= pid_max)
>>>>- pid = RESERVED_PIDS;
>>>>+ if (!pid_l) {
>>>>+ pid = last + 1;
>>>>+ if (pid >= pid_max)
>>>>+ pid = RESERVED_PIDS;
>>>>+ } else {
>>>>+ /*
>>>>+ * There's a target pid, so use it instead
>>>>+ */
>>>>+ BUG_ON(level < 0);
>>>>+ pid = PID_AT(pid_l, level);
>>>>+ if (pid >= pid_max)
>>>>+ return -EINVAL;
>>>>+ }
>>>>+
>>>> offset = pid & BITS_PER_PAGE_MASK;
>>>> map = &pid_ns->pidmap[pid/BITS_PER_PAGE];
>>>> max_scan = (pid_max + BITS_PER_PAGE - 1)/BITS_PER_PAGE - !offset;
>>>>@@ -153,9 +165,16 @@ static int alloc_pidmap(struct pid_names
>>>> do {
>>>> if (!test_and_set_bit(offset, map->page)) {
>>>> atomic_dec(&map->nr_free);
>>>>- pid_ns->last_pid = pid;
>>>>+ if (!pid_l)

```
>>>>+      pid_ns->last_pid = pid;
>>>>+     else
>>>>+      pid_ns->last_pid = max(last,
>>>>+         pid);
>>>>     return pid;
>>>>    }
>>>>+    if (pid_l)
>>>>+     /* Target pid is already in use */
>>>>+     return -EBUSY;
>>>>    offset = find_next_offset(map, offset);
>>>>    pid = mk_pid(pid_ns, map, offset);
>>>>   /*
>>>>@@ -179,7 +198,7 @@ static int alloc_pidmap(struct pid_names
>>>>  }
>>>>  pid = mk_pid(pid_ns, map, offset);
>>>> }
>>>>- return -1;
>>>>+ return -ENOMEM;
>>>>}
>>>>
>>>>int next_pidmap(struct pid_namespace *pid_ns, int last)
>>>
>>>As fas as this particular piece of code is concerned this all can
>>>be shrunk down to
>>>
>>>static int set_vpidmap(struct pid_namespace *ns, int pid)
>>>{
>>>        int offset;
>>>        pidmap_t *map;
>>>
>>>        offset = pid & BITS_PER_PAGE_MASK;
>>>        map = ns->pidmap + vpid / BITS_PER_PAGE;
>>>
>>>        if (unlikely(alloc_pidmap_page(map)))
>>>                return -ENOMEM;
>>>
>>>        if (test_and_set_bit(offset, map->page))
>>>                return -EEXIST;
>>>
>>>        atomic_dec(&map->nr_free);
>>>        return pid;
>>>}
>>>
>>>where the alloc_pidmap_page is a consolidated part of code from alloc_pidmap.
>>>
>>>And I'm scared of what the alloc_pid is going to become.
>>>
>>>
```

```
>>
>>It's true that I made alloc_pid() become uggly, but this patchset was
>>more intended to continue a discussion.
>>
>>What we could do is the following (not compiled, not tested...):
>>
>>struct pid *alloc_pid(struct pid_namespace *ns)
>>{
>>        struct pid *pid;
>>        enum pid_type type;
>>        int i, nr;
>>        struct pid_namespace *tmp;
>>        struct upid *upid;
>>
>>        pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
>>        if (!pid) {
>>                pid = ERR_PTR(-ENOMEM);
>>                goto out;
>>        }
>>
>>        tmp = ns;
>>        i = ns->level;
>>
>>        if (current->next_id && (current->next_id->flag & SYS_ID_PID)) {
>>                tmp = set_predefined_pids(ns,
>>                                current->next_id->pid_ids);
>>                if (IS_ERR(tmp)) {
>>                        nr = PTR_ERR(tmp);
>>                        goto out_free;
>>                }
>>        }
>>
>>        /*
>>         * Let the lower levels upid nrs be automatically allocated
>>         */
>>        for ( ; i >= 0; i--) {
>>                nr = alloc_pidmap(tmp, NULL, -1);
>>                if (nr < 0)
>>                        goto out_free;
>>....
>>
>>which would only add a test and a function call to alloc_pid() ==> more
>>readable.
>>with set_predefined_pids defined as follows (still not compiled, not
>>tested, ...):
>>
>>struct pid_namespace *set_predefined_pids(struct pid_namespace *ns,
>>                                        struct pid_list *pid_l)
```

```
>>{
>>        int rel_level;
>>
>>        BUG_ON(!pid_l);
>>
>>        rel_level = pid_l->npids - 1;
>>        if (rel_level > ns->level)
>>                return ERR_PTR(-EINVAL);
>>
>>        /*
>>         * Use the predefined upid nrs for levels ns->level down to
>>         * ns->level - rel_level
>>         */
>>        for ( ; rel_level >= 0; i--, rel_level--) {
>>                nr = alloc_pidmap(tmp, pid_l, rel_level);
>>                if (nr < 0)
>>                        return ERR_PTR(nr);
>>
>>                pid->numbers[i].nr = nr;
>>                pid->numbers[i].ns = tmp;
>>                tmp = tmp->parent;
>>        }
>>
>>        current->next_id->flag &= ~SYS_ID_PID;
>>        pids_free(pid_l);
>>        current->next_id->pid_ids = NULL;
>>
>>        return tmp;
>>}
>>
>>
>>Don't you think that mixing this with your 1st proposal (the
>>set_vpidmap() one), would make things look better?
>
>
> I'd prefer seeing
>
> --- a/kernel/pid.c
> +++ b/kernel/pid.c
> @@ -247,7 +247,7 @@ struct pid *alloc_pid(struct pid_namespace *ns)
>  {
>         struct pid *pid;
>         enum pid_type type;
> -       int i, nr;
> +       int i, nr, req_nr;
>         struct pid_namespace *tmp;
>         struct upid *upid;
>
```

> @@ -257,7 +257,11 @@ struct pid *alloc_pid(struct pid_namespace *ns)
>
> 	tmp = ns;
> 	for (i = ns->level; i >= 0; i--) {
> - 		nr = alloc_pidmap(tmp);
> + 		req_nr = get_required_pidnr(ns, i);
> + 		if (req_nr > 0)
> + 			nr = set_pidmap(tmp, req_nr);
> + 		else
> + 			nr = alloc_pidmap(tmp);
> 		if (nr < 0)
> 			goto out_free;
>
>
> in alloc_pid() and not much than that.
>
>

But doing this in the existing path, we are unconditionally adding for
each level:
. a call to get_required_pidnr()
. a test (on req_nr value)

while with what I proposed in my previous mail I'm adding only 1 test to
the existing alloc_pid() path. But I agree with that it may be still not
perfect.

Or may be I missed something?

Regards,
Nadia

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

---

Subject: Re: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by serue on Tue, 11 Mar 2008 16:47:25 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> Nadia Derbey wrote:
> > Pavel Emelyanov wrote:
> >> Nadia.Derbey@bull.net wrote:
> >>
> >>> @@ -122,14 +122,26 @@ static void free_pidmap(struct upid *upi
> >>> atomic_inc(&map->nr_free);

```
> >>> }
> >>>
> >>> -static int alloc_pidmap(struct pid_namespace *pid_ns)
> >>> +static int alloc_pidmap(struct pid_namespace *pid_ns, struct pid_list *pid_l,
> >>> +   int level)
> >>> {
> >>>  int i, offset, max_scan, pid, last = pid_ns->last_pid;
> >>>  struct pidmap *map;
> >>>
> >>> - pid = last + 1;
> >>> - if (pid >= pid_max)
> >>> -  pid = RESERVED_PIDS;
> >>> + if (!pid_l) {
> >>> +  pid = last + 1;
> >>> +  if (pid >= pid_max)
> >>> +   pid = RESERVED_PIDS;
> >>> + } else {
> >>> +  /*
> >>> +   * There's a target pid, so use it instead
> >>> +   */
> >>> +  BUG_ON(level < 0);
> >>> +  pid = PID_AT(pid_l, level);
> >>> +  if (pid >= pid_max)
> >>> +   return -EINVAL;
> >>> + }
> >>> +
> >>>  offset = pid & BITS_PER_PAGE_MASK;
> >>>  map = &pid_ns->pidmap[pid/BITS_PER_PAGE];
> >>>  max_scan = (pid_max + BITS_PER_PAGE - 1)/BITS_PER_PAGE - !offset;
> >>> @@ -153,9 +165,16 @@ static int alloc_pidmap(struct pid_names
> >>>    do {
> >>>     if (!test_and_set_bit(offset, map->page)) {
> >>>      atomic_dec(&map->nr_free);
> >>> -     pid_ns->last_pid = pid;
> >>> +     if (!pid_l)
> >>> +      pid_ns->last_pid = pid;
> >>> +     else
> >>> +      pid_ns->last_pid = max(last,
> >>> +         pid);
> >>>      return pid;
> >>>     }
> >>> +   if (pid_l)
> >>> +    /* Target pid is already in use */
> >>> +    return -EBUSY;
> >>>     offset = find_next_offset(map, offset);
> >>>     pid = mk_pid(pid_ns, map, offset);
> >>>    /*
> >>> @@ -179,7 +198,7 @@ static int alloc_pidmap(struct pid_names
```

> >>> }
> >>> pid = mk_pid(pid_ns, map, offset);
> >>> }
> >>> - return -1;
> >>> + return -ENOMEM;
> >>> }
> >>>
> >>> int next_pidmap(struct pid_namespace *pid_ns, int last)
> >>
> >> As fas as this particular piece of code is concerned this all can
> >> be shrunk down to
> >>
> >> static int set_vpidmap(struct pid_namespace *ns, int pid)
> >> {
> >>   int offset;
> >>   pidmap_t *map;
> >>
> >>   offset = pid & BITS_PER_PAGE_MASK;
> >>   map = ns->pidmap + vpid / BITS_PER_PAGE;
> >>
> >>   if (unlikely(alloc_pidmap_page(map)))
> >>   return -ENOMEM;
> >>
> >>   if (test_and_set_bit(offset, map->page))
> >>   return -EEXIST;
> >>
> >>   atomic_dec(&map->nr_free);
> >>   return pid;
> >> }
> >>
> >> where the alloc_pidmap_page is a consolidated part of code from alloc_pidmap.
> >>
> >> And I'm scared of what the alloc_pid is going to become.
> >>
> >>
> >
> > It's true that I made alloc_pid() become uggly, but this patchset was
> > more intended to continue a discussion.
> >
> > What we could do is the following (not compiled, not tested...):
> >
> > struct pid *alloc_pid(struct pid_namespace *ns)
> > {
> >   struct pid *pid;
> >   enum pid_type type;
> >   int i, nr;
> >   struct pid_namespace *tmp;
> >   struct upid *upid;

```
> >
> >         pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
> >         if (!pid) {
> >                 pid = ERR_PTR(-ENOMEM);
> >                 goto out;
> >         }
> >
> >         tmp = ns;
> >         i = ns->level;
> >
> >         if (current->next_id && (current->next_id->flag & SYS_ID_PID)) {
> >                 tmp = set_predefined_pids(ns,
> >                                 current->next_id->pid_ids);
> >                 if (IS_ERR(tmp)) {
> >                         nr = PTR_ERR(tmp);
> >                         goto out_free;
> >                 }
> >         }
> >
> >         /*
> >          * Let the lower levels upid nrs be automatically allocated
> >          */
> >         for ( ; i >= 0; i--) {
> >                 nr = alloc_pidmap(tmp, NULL, -1);
> >                 if (nr < 0)
> >                         goto out_free;
> > ....
> >
> > which would only add a test and a function call to alloc_pid() ==> more
> > readable.
> > with set_predefined_pids defined as follows (still not compiled, not
> > tested, ...):
> >
> > struct pid_namespace *set_predefined_pids(struct pid_namespace *ns,
> >                                           struct pid_list *pid_l)
> > {
> >         int rel_level;
> >
> >         BUG_ON(!pid_l);
> >
> >         rel_level = pid_l->npids - 1;
> >         if (rel_level > ns->level)
> >                 return ERR_PTR(-EINVAL);
> >
> >         /*
> >          * Use the predefined upid nrs for levels ns->level down to
> >          * ns->level - rel_level
> >          */
```

```
> >         for ( ; rel_level >= 0; i--, rel_level--) {
> >                 nr = alloc_pidmap(tmp, pid_l, rel_level);
> >                 if (nr < 0)
> >                         return ERR_PTR(nr);
> >
> >                 pid->numbers[i].nr = nr;
> >                 pid->numbers[i].ns = tmp;
> >                 tmp = tmp->parent;
> >         }
> >
> >         current->next_id->flag &= ~SYS_ID_PID;
> >         pids_free(pid_l);
> >         current->next_id->pid_ids = NULL;
> >
> >         return tmp;
> > }
> >
> >
> > Don't you think that mixing this with your 1st proposal (the
> > set_vpidmap() one), would make things look better?
>
> I'd prefer seeing
>
> --- a/kernel/pid.c
> +++ b/kernel/pid.c
> @@ -247,7 +247,7 @@ struct pid *alloc_pid(struct pid_namespace *ns)
>  {
>         struct pid *pid;
>         enum pid_type type;
> -       int i, nr;
> +       int i, nr, req_nr;
>         struct pid_namespace *tmp;
>         struct upid *upid;
>
> @@ -257,7 +257,11 @@ struct pid *alloc_pid(struct pid_namespace *ns)
>
>         tmp = ns;
>         for (i = ns->level; i >= 0; i--) {
> -               nr = alloc_pidmap(tmp);
> +               req_nr = get_required_pidnr(ns, i);
> +               if (req_nr > 0)
> +                       nr = set_pidmap(tmp, req_nr);
```

I assume you mean set_vpidmap(tmp, req_nr); here?

```
> +               else
> +                       nr = alloc_pidmap(tmp);
>                 if (nr < 0)
```

> goto out_free;
>
>
> in alloc_pid() and not much than that.

So get_required_pidnr(ns, i) would do something like

```
int get_required_pidnr(struct pid_namespace *ns, int i)
{
 if (current->next_id && (current->next_id->flag & SYS_ID_PID)) {
  pid_l = current->next_id->pid_ids;
  if (!pid_l) return 0;
  rel_level = pid_l->npids - 1;
  if (rel_level <= i)
   return PID_AT(pid_l, i);
 }
 return 0;
}
```

?

>
> > 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

---

Subject: Re: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by Pavel Emelianov on Tue, 11 Mar 2008 16:55:44 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelyanov (xemul@openvz.org):
>> Nadia Derbey wrote:
>>> Pavel Emelyanov wrote:
>>>> Nadia.Derbey@bull.net wrote:
>>>>
>>>>> @@ -122,14 +122,26 @@ static void free_pidmap(struct upid *upi
>>>>> atomic_inc(&map->nr_free);

```
>>>>> }
>>>>>
>>>>> -static int alloc_pidmap(struct pid_namespace *pid_ns)
>>>>> +static int alloc_pidmap(struct pid_namespace *pid_ns, struct pid_list *pid_l,
>>>>> +   int level)
>>>>> {
>>>>>  int i, offset, max_scan, pid, last = pid_ns->last_pid;
>>>>>  struct pidmap *map;
>>>>>
>>>>> - pid = last + 1;
>>>>> - if (pid >= pid_max)
>>>>> -  pid = RESERVED_PIDS;
>>>>> + if (!pid_l) {
>>>>> +  pid = last + 1;
>>>>> +  if (pid >= pid_max)
>>>>> +   pid = RESERVED_PIDS;
>>>>> + } else {
>>>>> +  /*
>>>>> +   * There's a target pid, so use it instead
>>>>> +   */
>>>>> +  BUG_ON(level < 0);
>>>>> +  pid = PID_AT(pid_l, level);
>>>>> +  if (pid >= pid_max)
>>>>> +   return -EINVAL;
>>>>> + }
>>>>> +
>>>>>  offset = pid & BITS_PER_PAGE_MASK;
>>>>>  map = &pid_ns->pidmap[pid/BITS_PER_PAGE];
>>>>>  max_scan = (pid_max + BITS_PER_PAGE - 1)/BITS_PER_PAGE - !offset;
>>>>> @@ -153,9 +165,16 @@ static int alloc_pidmap(struct pid_names
>>>>>    do {
>>>>>     if (!test_and_set_bit(offset, map->page)) {
>>>>>      atomic_dec(&map->nr_free);
>>>>> -     pid_ns->last_pid = pid;
>>>>> +     if (!pid_l)
>>>>> +      pid_ns->last_pid = pid;
>>>>> +     else
>>>>> +      pid_ns->last_pid = max(last,
>>>>> +         pid);
>>>>>      return pid;
>>>>>     }
>>>>> +   if (pid_l)
>>>>> +    /* Target pid is already in use */
>>>>> +    return -EBUSY;
>>>>>     offset = find_next_offset(map, offset);
>>>>>     pid = mk_pid(pid_ns, map, offset);
>>>>>    /*
>>>>> @@ -179,7 +198,7 @@ static int alloc_pidmap(struct pid_names
```

```
>>>>>   }
>>>>>   pid = mk_pid(pid_ns, map, offset);
>>>>>  }
>>>>> - return -1;
>>>>> + return -ENOMEM;
>>>>> }
>>>>>
>>>>> int next_pidmap(struct pid_namespace *pid_ns, int last)
>>>> As fas as this particular piece of code is concerned this all can
>>>> be shrunk down to
>>>>
>>>> static int set_vpidmap(struct pid_namespace *ns, int pid)
>>>> {
>>>>         int offset;
>>>>         pidmap_t *map;
>>>>
>>>>         offset = pid & BITS_PER_PAGE_MASK;
>>>>         map = ns->pidmap + vpid / BITS_PER_PAGE;
>>>>
>>>>         if (unlikely(alloc_pidmap_page(map)))
>>>>                 return -ENOMEM;
>>>>
>>>>         if (test_and_set_bit(offset, map->page))
>>>>                 return -EEXIST;
>>>>
>>>>         atomic_dec(&map->nr_free);
>>>>         return pid;
>>>> }
>>>>
>>>> where the alloc_pidmap_page is a consolidated part of code from alloc_pidmap.
>>>>
>>>> And I'm scared of what the alloc_pid is going to become.
>>>>
>>>>
>>> It's true that I made alloc_pid() become uggly, but this patchset was
>>> more intended to continue a discussion.
>>>
>>> What we could do is the following (not compiled, not tested...):
>>>
>>> struct pid *alloc_pid(struct pid_namespace *ns)
>>> {
>>>         struct pid *pid;
>>>         enum pid_type type;
>>>         int i, nr;
>>>         struct pid_namespace *tmp;
>>>         struct upid *upid;
>>>
>>>         pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
```

```
>>>         if (!pid) {
>>>                 pid = ERR_PTR(-ENOMEM);
>>>                 goto out;
>>>         }
>>>
>>>         tmp = ns;
>>>         i = ns->level;
>>>
>>>         if (current->next_id && (current->next_id->flag & SYS_ID_PID)) {
>>>                 tmp = set_predefined_pids(ns,
>>>                                 current->next_id->pid_ids);
>>>                 if (IS_ERR(tmp)) {
>>>                         nr = PTR_ERR(tmp);
>>>                         goto out_free;
>>>                 }
>>>         }
>>>
>>>         /*
>>>          * Let the lower levels upid nrs be automatically allocated
>>>          */
>>>         for ( ; i >= 0; i--) {
>>>                 nr = alloc_pidmap(tmp, NULL, -1);
>>>                 if (nr < 0)
>>>                         goto out_free;
>>> ....
>>>
>>> which would only add a test and a function call to alloc_pid() ==> more
>>> readable.
>>> with set_predefined_pids defined as follows (still not compiled, not
>>> tested, ...):
>>>
>>> struct pid_namespace *set_predefined_pids(struct pid_namespace *ns,
>>>                                           struct pid_list *pid_l)
>>> {
>>>         int rel_level;
>>>
>>>         BUG_ON(!pid_l);
>>>
>>>         rel_level = pid_l->npids - 1;
>>>         if (rel_level > ns->level)
>>>                 return ERR_PTR(-EINVAL);
>>>
>>>         /*
>>>          * Use the predefined upid nrs for levels ns->level down to
>>>          * ns->level - rel_level
>>>          */
>>>         for ( ; rel_level >= 0; i--, rel_level--) {
>>>                 nr = alloc_pidmap(tmp, pid_l, rel_level);
```

```
>>>                 if (nr < 0)
>>>                         return ERR_PTR(nr);
>>>
>>>                 pid->numbers[i].nr = nr;
>>>                 pid->numbers[i].ns = tmp;
>>>                 tmp = tmp->parent;
>>>         }
>>>
>>>         current->next_id->flag &= ~SYS_ID_PID;
>>>         pids_free(pid_l);
>>>         current->next_id->pid_ids = NULL;
>>>
>>>         return tmp;
>>> }
>>>
>>>
>>> Don't you think that mixing this with your 1st proposal (the
>>> set_vpidmap() one), would make things look better?
>> I'd prefer seeing
>>
>> --- a/kernel/pid.c
>> +++ b/kernel/pid.c
>> @@ -247,7 +247,7 @@ struct pid *alloc_pid(struct pid_namespace *ns)
>>  {
>>         struct pid *pid;
>>         enum pid_type type;
>> -       int i, nr;
>> +       int i, nr, req_nr;
>>         struct pid_namespace *tmp;
>>         struct upid *upid;
>>
>> @@ -257,7 +257,11 @@ struct pid *alloc_pid(struct pid_namespace *ns)
>>
>>         tmp = ns;
>>         for (i = ns->level; i >= 0; i--) {
>> -               nr = alloc_pidmap(tmp);
>> +               req_nr = get_required_pidnr(ns, i);
>> +               if (req_nr > 0)
>> +                       nr = set_pidmap(tmp, req_nr);
>
> I assume you mean set_vpidmap(tmp, req_nr); here?

:) or however this one is called.

>> +               else
>> +                       nr = alloc_pidmap(tmp);
>>                 if (nr < 0)
>>                         goto out_free;
```

>>
>>
>> in alloc_pid() and not much than that.
>
> So get_required_pidnr(ns, i) would do something like
>
> int get_required_pidnr(struct pid_namespace *ns, int i)
> {
> if (current->next_id && (current->next_id->flag & SYS_ID_PID)) {
>   pid_l = current->next_id->pid_ids;
>   if (!pid_l) return 0;
>   rel_level = pid_l->npids - 1;
>   if (rel_level <= i)
>   return PID_AT(pid_l, i);
> }
> return 0;
> }
>
> ?

Well, yes, sort of. I haven't looked close to this part of patch,
but looks correct.

>>> 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

---

Subject: Re: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by serue on Tue, 11 Mar 2008 17:53:28 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > Quoting Pavel Emelyanov (xemul@openvz.org):
> >> Nadia Derbey wrote:
> >>> Pavel Emelyanov wrote:
> >>>> Nadia.Derbey@bull.net wrote:

```
> >>>>
> >>>>> @@ -122,14 +122,26 @@ static void free_pidmap(struct upid *upi
> >>>>>  atomic_inc(&map->nr_free);
> >>>>> }
> >>>>>
> >>>>> -static int alloc_pidmap(struct pid_namespace *pid_ns)
> >>>>> +static int alloc_pidmap(struct pid_namespace *pid_ns, struct pid_list *pid_l,
> >>>>> +   int level)
> >>>>> {
> >>>>>  int i, offset, max_scan, pid, last = pid_ns->last_pid;
> >>>>>  struct pidmap *map;
> >>>>>
> >>>>> - pid = last + 1;
> >>>>> - if (pid >= pid_max)
> >>>>> -  pid = RESERVED_PIDS;
> >>>>> + if (!pid_l) {
> >>>>> +  pid = last + 1;
> >>>>> +  if (pid >= pid_max)
> >>>>> +   pid = RESERVED_PIDS;
> >>>>> + } else {
> >>>>> +  /*
> >>>>> +   * There's a target pid, so use it instead
> >>>>> +   */
> >>>>> +  BUG_ON(level < 0);
> >>>>> +  pid = PID_AT(pid_l, level);
> >>>>> +  if (pid >= pid_max)
> >>>>> +   return -EINVAL;
> >>>>> + }
> >>>>> +
> >>>>>  offset = pid & BITS_PER_PAGE_MASK;
> >>>>>  map = &pid_ns->pidmap[pid/BITS_PER_PAGE];
> >>>>>  max_scan = (pid_max + BITS_PER_PAGE - 1)/BITS_PER_PAGE - !offset;
> >>>>> @@ -153,9 +165,16 @@ static int alloc_pidmap(struct pid_names
> >>>>>    do {
> >>>>>     if (!test_and_set_bit(offset, map->page)) {
> >>>>>      atomic_dec(&map->nr_free);
> >>>>> -     pid_ns->last_pid = pid;
> >>>>> +     if (!pid_l)
> >>>>> +      pid_ns->last_pid = pid;
> >>>>> +     else
> >>>>> +      pid_ns->last_pid = max(last,
> >>>>> +         pid);
> >>>>>      return pid;
> >>>>>     }
> >>>>> +   if (pid_l)
> >>>>> +    /* Target pid is already in use */
> >>>>> +    return -EBUSY;
> >>>>>     offset = find_next_offset(map, offset);
```

```
> >>>>>     pid = mk_pid(pid_ns, map, offset);
> >>>>>    /*
> >>>>> @@ -179,7 +198,7 @@ static int alloc_pidmap(struct pid_names
> >>>>>   }
> >>>>>   pid = mk_pid(pid_ns, map, offset);
> >>>>>  }
> >>>>> - return -1;
> >>>>> + return -ENOMEM;
> >>>>> }
> >>>>>
> >>>>> int next_pidmap(struct pid_namespace *pid_ns, int last)
> >>>> As fas as this particular piece of code is concerned this all can
> >>>> be shrunk down to
> >>>>
> >>>> static int set_vpidmap(struct pid_namespace *ns, int pid)
> >>>> {
> >>>>         int offset;
> >>>>         pidmap_t *map;
> >>>>
> >>>>         offset = pid & BITS_PER_PAGE_MASK;
> >>>>         map = ns->pidmap + vpid / BITS_PER_PAGE;
> >>>>
> >>>>         if (unlikely(alloc_pidmap_page(map)))
> >>>>                 return -ENOMEM;
> >>>>
> >>>>         if (test_and_set_bit(offset, map->page))
> >>>>                 return -EEXIST;
> >>>>
> >>>>         atomic_dec(&map->nr_free);
> >>>>         return pid;
> >>>> }
> >>>>
> >>>> where the alloc_pidmap_page is a consolidated part of code from alloc_pidmap.
> >>>>
> >>>> And I'm scared of what the alloc_pid is going to become.
> >>>>
> >>>>
> >>> It's true that I made alloc_pid() become uggly, but this patchset was
> >>> more intended to continue a discussion.
> >>>
> >>> What we could do is the following (not compiled, not tested...):
> >>>
> >>> struct pid *alloc_pid(struct pid_namespace *ns)
> >>> {
> >>>         struct pid *pid;
> >>>         enum pid_type type;
> >>>         int i, nr;
> >>>         struct pid_namespace *tmp;
```

```
> >>>         struct upid *upid;
> >>>
> >>>         pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
> >>>         if (!pid) {
> >>>                 pid = ERR_PTR(-ENOMEM);
> >>>                 goto out;
> >>>         }
> >>>
> >>>         tmp = ns;
> >>>         i = ns->level;
> >>>
> >>>         if (current->next_id && (current->next_id->flag & SYS_ID_PID)) {
> >>>                 tmp = set_predefined_pids(ns,
> >>>                                 current->next_id->pid_ids);
> >>>                 if (IS_ERR(tmp)) {
> >>>                         nr = PTR_ERR(tmp);
> >>>                         goto out_free;
> >>>                 }
> >>>         }
> >>>
> >>>         /*
> >>>          * Let the lower levels upid nrs be automatically allocated
> >>>          */
> >>>         for ( ; i >= 0; i--) {
> >>>                 nr = alloc_pidmap(tmp, NULL, -1);
> >>>                 if (nr < 0)
> >>>                         goto out_free;
> >>> ....
> >>>
> >>> which would only add a test and a function call to alloc_pid() ==> more
> >>> readable.
> >>> with set_predefined_pids defined as follows (still not compiled, not
> >>> tested, ...):
> >>>
> >>> struct pid_namespace *set_predefined_pids(struct pid_namespace *ns,
> >>>                                           struct pid_list *pid_l)
> >>> {
> >>>         int rel_level;
> >>>
> >>>         BUG_ON(!pid_l);
> >>>
> >>>         rel_level = pid_l->npids - 1;
> >>>         if (rel_level > ns->level)
> >>>                 return ERR_PTR(-EINVAL);
> >>>
> >>>         /*
> >>>          * Use the predefined upid nrs for levels ns->level down to
> >>>          * ns->level - rel_level
```

```
> >>>          */
> >>>         for ( ; rel_level >= 0; i--, rel_level--) {
> >>>                 nr = alloc_pidmap(tmp, pid_l, rel_level);
> >>>                 if (nr < 0)
> >>>                         return ERR_PTR(nr);
> >>>
> >>>                 pid->numbers[i].nr = nr;
> >>>                 pid->numbers[i].ns = tmp;
> >>>                 tmp = tmp->parent;
> >>>         }
> >>>
> >>>         current->next_id->flag &= ~SYS_ID_PID;
> >>>         pids_free(pid_l);
> >>>         current->next_id->pid_ids = NULL;
> >>>
> >>>         return tmp;
> >>> }
> >>>
> >>>
> >>> Don't you think that mixing this with your 1st proposal (the
> >>> set_vpidmap() one), would make things look better?
> >> I'd prefer seeing
> >>
> >> --- a/kernel/pid.c
> >> +++ b/kernel/pid.c
> >> @@ -247,7 +247,7 @@ struct pid *alloc_pid(struct pid_namespace *ns)
> >>  {
> >>         struct pid *pid;
> >>         enum pid_type type;
> >> -       int i, nr;
> >> +       int i, nr, req_nr;
> >>         struct pid_namespace *tmp;
> >>         struct upid *upid;
> >>
> >> @@ -257,7 +257,11 @@ struct pid *alloc_pid(struct pid_namespace *ns)
> >>
> >>         tmp = ns;
> >>         for (i = ns->level; i >= 0; i--) {
> >> -               nr = alloc_pidmap(tmp);
> >> +               req_nr = get_required_pidnr(ns, i);
> >> +               if (req_nr > 0)
> >> +                       nr = set_pidmap(tmp, req_nr);
> > I assume you mean set_vpidmap(tmp, req_nr); here?
>
> :) or however this one is called.
```

Right, I just meant did you mean the same fn as in your previous

msg :) Don't care what it's called, actuall set_pidmap probably
is better, as 'virtual' isn't quite right.

> >> + else
> >> + nr = alloc_pidmap(tmp);
> >> if (nr < 0)
> >> goto out_free;
> >>
> >>
> >> in alloc_pid() and not much than that.
> >
> > So get_required_pidnr(ns, i) would do something like
> >
> > int get_required_pidnr(struct pid_namespace *ns, int i)
> > {
> > if (current->next_id && (current->next_id->flag & SYS_ID_PID)) {
> > pid_l = current->next_id->pid_ids;
> > if (!pid_l) return 0;
> > rel_level = pid_l->npids - 1;
> > if (rel_level <= i)
> > return PID_AT(pid_l, i);
> > }
> > return 0;
> > }
> >
> > ?
>
> Well, yes, sort of. I haven't looked close to this part of patch,
> but looks correct.

Right the alloc_pidmap() changes will probably be pretty much the same
no matter how we do set_it(), so it's worth discussing. But I'm
particularly curious to see what opinions are on the sys_setid().

-serge

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

---

Subject: Re: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by ebiederm on Wed, 12 Mar 2008 19:58:56 GMT
View Forum Message <> Reply to Message

"Nadia Derbey" <Nadia.Derbey@bull.net> writes:
> A couple of weeks ago, a discussion has started after Pierre's proposal for
> a new syscall to change an ipc id (see thread

> http://lkml.org/lkml/2008/1/29/209).
>
>
> Oren's suggestion was to force an object's id during its creation, rather
> than 1. create it, 2. change its id.
>
> So here is an implementation of what Oren has suggested.
>
> 2 new files are defined under /proc/self:
> . next_ipcid --> next id to use for ipc object creation
> . next_pids --> next upid nr(s) to use for next task to be forked
> (see patch #2 for more details).
>
> When one of these files (or both of them) is filled, a structure pointed to
> by the calling task struct is filled with these ids.
>
> Then, when the object is created, the id(s) present in that structure are
> used, instead of the default ones.
> A couple of weeks ago, a discussion has started after Pierre's proposal for
> a new syscall to change an ipc id (see thread
> http://lkml.org/lkml/2008/1/29/209).
>
>
> Oren's suggestion was to force an object's id during its creation, rather
> than 1. create it, 2. change its id.
>
> So here is an implementation of what Oren has suggested.
>
> 2 new files are defined under /proc/self:
> . next_ipcid --> next id to use for ipc object creation
> . next_pids --> next upid nr(s) to use for next task to be forked
> (see patch #2 for more details).
>
> When one of these files (or both of them) is filled, a structure pointed to
> by the calling task struct is filled with these ids.
>
> Then, when the object is created, the id(s) present in that structure are
> used, instead of the default ones.

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Right the alloc_pidmap() changes will probably be pretty much the same
> no matter how we do set_it(), so it's worth discussing. But I'm
> particularly curious to see what opinions are on the sys_setid().

A couple of comments. With respect to alloc_pidmap we already have
the necessary controls (a minimum and a maximum) in place for the
allocation. So except for double checking that those controls are exported

in /proc/sys we don't necessarily need to do anything, special.

Just play games with the minimum pid value before you fork.

Second at least to get the memory map correct we need additional
kernel support.

Third to actually get the values out it appears we need additional kernel
support as well. From my limited playing with these things at least
parts of the code were easier to implement in the kernel. The hoops
you have to go to restore a single process (without threads) are
absolutely horrendous in user space.

So this patchset whether it is setid or setting the id at creation time
seems to be jumping the gun. For some namespaces renames are valid and
we can support them. For other namespaces setting the id is a big no-no,
and possibly even controlling the id at creation time is a problem (at
least in the general sense). Because if you can easily control the id
you may be able to more easily exploit security holes.

I'm not at all inclined to make it easy for userspace to rename or set
the id of a new resource unless it already makes sense in that
namespace.

We need to limit anything in a checkpoint to user space visible
state. For sockets this can get fairly abstract since on the wire
state of a tcp socket is in some sense user visible. However it
should not include things that user space can never see like socket
hash values.

Partly what this set of patches demonstrates is that it is fairly
straight forward to restore ids. Getting the little details of the
proper maintainable user space interface correct is harder.

The goal with any userspace implementation is to that we can separate
policy form mechanism. So what are the trade offs for various
approaches.

At least for inspection at the checkpoint side it would be nice for
debugging applications to easily get at all of the user space visible
state. So there is an argument for making state that is only
indirectly visible, visible for diagnostic and debugging purposes.

For saving the state to disk it appears we need to stop all of the

processes in our container. Again something a debugging application
of an entire container may want to do. Although we also want to stop
the hardware queues for things like networking so we don't transmit
or possibly receive new packets either.

We want a checkpoint/restart to be essentially atomic, with non
of the tasks that we stop being able to prove that they ran while
the checkpoint was being taken. Mostly this is an interprocess
communication blackout, but there may be more to it then that.

We want checkpoint/restart if possible to be incremental. So
we can perform actions like live migration efficiently if most of
the data is not changing.

So there is an argument to perform the work piecemeal instead of
in one big shot. Although if we can load data from wherever
into the kernel data structures quickly it may not be a big deal.

We also want the transfer of state to be fast. Which tends to argue
in the other direction. That we want a bulk operation that can save
out everything and restore everything quickly.

We also want a design that we can implement incrementally. So that
we can avoid supporting everything at first and if there is state
that we should save that we can't (or similarly state that we should
restore but we can't) the save/restore fails. Until that part is
implemented.

Further we need to finish difficult things like sysfs support and
proc support for simply running applications in containers.

So while I think it is good to be thinking and playing with these
ideas now. I think having a more complete story and not pecking on
the pieces right now is important.

My inclination is that create with a specified set of ids is the
proper internal kernel API, so we don't have rework things later,
because reworking things seems to be a lot more work. How we want to
export this to user space is another matter.

One suggestion is to a have /proc or a proc like filesystem that
allows us to read, create, and populate files to see all of the
application state. Then in userspace it is possible that the
transfer of the checkpoint would be as simple as rsync.

Ok back to my cave for a bit.

Eric

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

Subject: Re: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by Nadia Derbey on Thu, 13 Mar 2008 10:41:40 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> "Nadia Derbey" <Nadia.Derbey@bull.net> writes:
>
>>A couple of weeks ago, a discussion has started after Pierre's proposal for
>>a new syscall to change an ipc id (see thread
>>http://lkml.org/lkml/2008/1/29/209).
>>
>>
>>Oren's suggestion was to force an object's id during its creation, rather
>>than 1. create it, 2. change its id.
>>
>>So here is an implementation of what Oren has suggested.
>>
>>2 new files are defined under /proc/self:
>> . next_ipcid --> next id to use for ipc object creation
>> . next_pids --> next upid nr(s) to use for next task to be forked
>>   (see patch #2 for more details).
>>
>>When one of these files (or both of them) is filled, a structure pointed to
>>by the calling task struct is filled with these ids.
>>
>>Then, when the object is created, the id(s) present in that structure are
>>used, instead of the default ones.
>>A couple of weeks ago, a discussion has started after Pierre's proposal for
>>a new syscall to change an ipc id (see thread
>>http://lkml.org/lkml/2008/1/29/209).
>>
>>
>>Oren's suggestion was to force an object's id during its creation, rather
>>than 1. create it, 2. change its id.
>>
>>So here is an implementation of what Oren has suggested.
>>
>>2 new files are defined under /proc/self:
>> . next_ipcid --> next id to use for ipc object creation
>> . next_pids --> next upid nr(s) to use for next task to be forked
>>   (see patch #2 for more details).

>>
>>When one of these files (or both of them) is filled, a structure pointed to
>>by the calling task struct is filled with these ids.
>>
>>Then, when the object is created, the id(s) present in that structure are
>>used, instead of the default ones.
>
>
> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>
>
>>Right the alloc_pidmap() changes will probably be pretty much the same
>>no matter how we do set_it(), so it's worth discussing. But I'm
>>particularly curious to see what opinions are on the sys_setid().
>
>
> A couple of comments. With respect to alloc_pidmap we already have
> the necessary controls (a minimum and a maximum) in place for the
> allocation. So except for double checking that those controls are exported
> in /proc/sys we don't necessarily need to do anything, special.
>
> Just play games with the minimum pid value before you fork.

Excellent idea! It's trus that properly setting things, we can make the
loops executed only once in case we want to use a predefined id.

>
> Second at least to get the memory map correct we need additional
> kernel support.
>
>
> Third to actually get the values out it appears we need additional kernel
> support as well. From my limited playing with these things at least
> parts of the code were easier to implement in the kernel. The hoops
> you have to go to restore a single process (without threads) are
> absolutely horrendous in user space.

Ok, but if we have a process that belongs to nested namepsaces, we have
no other choice than provide its upid nrs hierarchy, right?
So, I guess it's more the way it is presented that you don't agree with
(of course provided that we are in a user space oriented solution)?

>
> So this patchset whether it is setid or setting the id at creation time
> seems to be jumping the gun. For some namespaces renames are valid and
> we can support them. For other namespaces setting the id is a big no-no,
> and possibly even controlling the id at creation time is a problem (at
> least in the general sense).

I'm sorry but I'm pretty new in this domain, so I don't see what are the
namespaces where setting (or pre-setting) the id would be a problem?

> Because if you can easily control the id
> you may be able to more easily exploit security holes.
>
> I'm not at all inclined to make it easy for userspace to rename or set
> the id of a new resource unless it already makes sense in that
> namespace.
>
> We need to limit anything in a checkpoint to user space visible
> state.

OK, but a tasks that belongs to nested pid namespaces is known from
other tasks as one of its "intermediate" pids, depending on the
namespace level we are considering. So we can consider these
"intermediate pids" as user space visible, can't we?

Given the following pid namespaces hierarchy:
PNS0 -> PNS1 -> PNS2 -> PNS3 -> PNS4
A task that belongs to PSN2 has 3 upid nrs:
UP0, UP1, and UP2
So:
. UP0 can be obtained if we do a "ps -ef" in pid ns #1 (PNS0)
. UP1 can be obtained if we do a "ps -ef" in pid ns #1 (PNS1)
. UP2 can be obtained if we do a "ps -ef" in pid ns #1 (PNS2)

So UP[0-2] are user space visible (again, depending on "where" we are in
pid ns hierarchy, aren't they?

> For sockets this can get fairly abstract since on the wire
> state of a tcp socket is in some sense user visible. However it
> should not include things that user space can never see like socket
> hash values.
>
> Partly what this set of patches demonstrates is that it is fairly
> straight forward to restore ids. Getting the little details of the
> proper maintainable user space interface correct is harder.
>
> The goal with any userspace implementation is to that we can separate
> policy form mechanism. So what are the trade offs for various
> approaches.
>
> At least for inspection at the checkpoint side it would be nice for
> debugging applications to easily get at all of the user space visible
> state. So there is an argument for making state that is only
> indirectly visible, visible for diagnostic and debugging purposes.

>
> For saving the state to disk it appears we need to stop all of the
> processes in our container. Again something a debugging application
> of an entire container may want to do. Although we also want to stop
> the hardware queues for things like networking so we don't transmit
> or possibly receive new packets either.
>
> We want a checkpoint/restart to be essentially atomic, with non
> of the tasks that we stop being able to prove that they ran while
> the checkpoint was being taken. Mostly this is an interprocess
> communication blackout, but there may be more to it then that.
>
> We want checkpoint/restart if possible to be incremental. So
> we can perform actions like live migration efficiently if most of
> the data is not changing.
>
> So there is an argument to perform the work piecemeal instead of
> in one big shot. Although if we can load data from wherever
> into the kernel data structures quickly it may not be a big deal.
>
> We also want the transfer of state to be fast. Which tends to argue
> in the other direction. That we want a bulk operation that can save
> out everything and restore everything quickly.
>
> We also want a design that we can implement incrementally. So that
> we can avoid supporting everything at first and if there is state
> that we should save that we can't (or similarly state that we should
> restore but we can't) the save/restore fails. Until that part is
> implemented.
>
> Further we need to finish difficult things like sysfs support and
> proc support for simply running applications in containers.
>
> So while I think it is good to be thinking and playing with these
> ideas now. I think having a more complete story and not pecking on
> the pieces right now is important.

Sure, I completely agree with you! So may be the 1st thing to do would
be to decide which approach (user space vs kernel) should be adopted for
c/r? Sorry if what I'm saying is stupid, but imho a clear answer to this
question would make us all go the same direction ==> save time for
further investigations.

>
> My inclination is that create with a specified set of ids is the
> proper internal kernel API, so we don't have rework things later,
> because reworking things seems to be a lot more work.

Completely agree with you: the earlier in the object's life we do it,
the best it is ;-)

> How we want to
> export this to user space is another matter.
>
> One suggestion is to a have /proc or a proc like filesystem that
> allows us to read, create, and populate files to see all of the
> application state.

And that would be quite useful for debugging purpose, as you were saying
earlier.

> Then in userspace it is possible that the
> transfer of the checkpoint would be as simple as rsync.
>
> Ok back to my cave for a bit.
>
> Eric
>
>
>

Regards,
Nadia

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

---

Subject: Re: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by ebiederm on Thu, 13 Mar 2008 17:40:01 GMT
View Forum Message <> Reply to Message

Nadia Derbey <Nadia.Derbey@bull.net> writes:

> Eric W. Biederman wrote:
>> "Nadia Derbey" <Nadia.Derbey@bull.net> writes:
>>
>>>A couple of weeks ago, a discussion has started after Pierre's proposal for
>>>a new syscall to change an ipc id (see thread
>>>http://lkml.org/lkml/2008/1/29/209).
>>>
>>>
>>>Oren's suggestion was to force an object's id during its creation, rather
>>>than 1. create it, 2. change its id.

>>>
>>>So here is an implementation of what Oren has suggested.
>>>
>>>2 new files are defined under /proc/self:
>>> . next_ipcid --> next id to use for ipc object creation
>>> . next_pids --> next upid nr(s) to use for next task to be forked
>>>   (see patch #2 for more details).
>>>
>>>When one of these files (or both of them) is filled, a structure pointed to
>>>by the calling task struct is filled with these ids.
>>>
>>>Then, when the object is created, the id(s) present in that structure are
>>>used, instead of the default ones.
>>>A couple of weeks ago, a discussion has started after Pierre's proposal for
>>>a new syscall to change an ipc id (see thread
>>>http://lkml.org/lkml/2008/1/29/209).
>>>
>>>
>>>Oren's suggestion was to force an object's id during its creation, rather
>>>than 1. create it, 2. change its id.
>>>
>>>So here is an implementation of what Oren has suggested.
>>>
>>>2 new files are defined under /proc/self:
>>> . next_ipcid --> next id to use for ipc object creation
>>> . next_pids --> next upid nr(s) to use for next task to be forked
>>>   (see patch #2 for more details).
>>>
>>>When one of these files (or both of them) is filled, a structure pointed to
>>>by the calling task struct is filled with these ids.
>>>
>>>Then, when the object is created, the id(s) present in that structure are
>>>used, instead of the default ones.
>>
>>
>> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>>
>>
>>>Right the alloc_pidmap() changes will probably be pretty much the same
>>>no matter how we do set_it(), so it's worth discussing. But I'm
>>>particularly curious to see what opinions are on the sys_setid().
>>
>>
>> A couple of comments. With respect to alloc_pidmap we already have
>> the necessary controls (a minimum and a maximum) in place for the
>> allocation. So except for double checking that those controls are exported
>> in /proc/sys we don't necessarily need to do anything, special.
>>

>> Just play games with the minimum pid value before you fork.
>
> Excellent idea! It's trus that properly setting things, we can make the loops
> executed only once in case we want to use a predefined id.
>
>>
>> Second at least to get the memory map correct we need additional
>> kernel support.
>>
>>
>> Third to actually get the values out it appears we need additional kernel
>> support as well. From my limited playing with these things at least
>> parts of the code were easier to implement in the kernel. The hoops
>> you have to go to restore a single process (without threads) are
>> absolutely horrendous in user space.
>

I was thinking in particular of the mm setup above.

> Ok, but if we have a process that belongs to nested namepsaces, we have no other
> choice than provide its upid nrs hierarchy, right?

At least a part of it. You only have to provide as much of the nested
pid hiearchy as you are restoring from your checkpoint.

> So, I guess it's more the way it is presented that you don't agree with (of
> course provided that we are in a user space oriented solution)?

Yes. On the kernel side we are essentially fine.

>> So this patchset whether it is setid or setting the id at creation time
>> seems to be jumping the gun. For some namespaces renames are valid and
>> we can support them. For other namespaces setting the id is a big no-no,
>> and possibly even controlling the id at creation time is a problem (at
>> least in the general sense).
>
> I'm sorry but I'm pretty new in this domain, so I don't see what are the
> namespaces where setting (or pre-setting) the id would be a problem?

pids to some extent as people use them in all kinds of files. Being
able to force the pid of another process could make a hard to trigger
security hole with file permissions absolutely trivial to hit.

Changing the pid on a process would be even worse because then how
could you send it signals.

I haven't looked close enough to be able to say in situation X it is a
problem or in situation Y it is clearly not a problem. I just know

there is a lot that happens with ids and security so we need to tread
lightly, and carefully.

>> Because if you can easily control the id
>> you may be able to more easily exploit security holes.
>>
>> I'm not at all inclined to make it easy for userspace to rename or set
>> the id of a new resource unless it already makes sense in that
>> namespace.
>>
>> We need to limit anything in a checkpoint to user space visible
>> state.
>
> OK, but a tasks that belongs to nested pid namespaces is known from other tasks
> as one of its "intermediate" pids, depending on the namespace level we are
> considering. So we can consider these "intermediate pids" as user space visible,
> can't we?

Yes.

> Given the following pid namespaces hierarchy:
> PNS0 -> PNS1 -> PNS2 -> PNS3 -> PNS4
> A task that belongs to PSN2 has 3 upid nrs:
> UP0, UP1, and UP2
> So:
> . UP0 can be obtained if we do a "ps -ef" in pid ns #1 (PNS0)
> . UP1 can be obtained if we do a "ps -ef" in pid ns #1 (PNS1)
> . UP2 can be obtained if we do a "ps -ef" in pid ns #1 (PNS2)
>
> So UP[0-2] are user space visible (again, depending on "where" we are in pid ns
> hierarchy, aren't they?

Totally.

> Sure, I completely agree with you! So may be the 1st thing to do would be to
> decide which approach (user space vs kernel) should be adopted for c/r? Sorry if
> what I'm saying is stupid, but imho a clear answer to this question would make
> us all go the same direction ==> save time for further investigations.

Agreed, although it isn't quite that cut and dry.

The question is what granularity do we export the checkpoint restore
functionality with, and do we manage to have it serve additional
functions as well.

Because ultimately it is user space saying checkpoint and it is user
space saying restore. Although we need to be careful that there
are not cheaper migration solutions that we are precluding.

Getting the user space interface correct is going to be the tricky
and important.

My inclination is that the cost in migrating a container is the cost
of moving the data between machines. Which (not counting filesystems)
would be the anonymous pages and the shared memory areas.

So if we have a checkpoint as a directory.
We could have files for the data of each shared memory region.

We could have files for the data of each shared anonymous region
shared between mm's.

We could have ELF core files with extra notes to describe the full
state of each process.

We could have files describing each sysvipc object (shared memory,
message queues, and semaphores).

Futexes?

I would suggest a directory for each different kind of file, making
conflicts easier to avoid, and data types easier to predict.

Once we have a least one valid checkpoint format the question becomes
how do we get the checkpoint out of the kernel, and how do we get
the checkpoint into the kernel. And especially how do we allow for
incremental data movement so live migration with minimal interruption
of service is possible.

This means that we need to preload all of the large memory areas into
shared memory areas or processes. Everything else file descriptors
and the like I expect will be sufficiently inexpesive that we can
treat their recreation as instantaneous.

If incremental migration of data was not so useful I would say just
have a single syscall to dump everything, and another syscall to
restore everything.

Since incremental migration is so useful. My gut feel is
checkpointfs or something similar where we can repeated read
and write a checkpoint before we commit to starting it is useful.

I am welcome to other better suggestions from the experience of the
people who have implemented this before.

A very fine grained user space solution where user space createsn
each object using the traditional APIs and restores them similarly
seems harder to implement, harder to get right, and harder to
maintain. In part because it hides what we are actually trying to do.

Further a fine grained user space approach appears to offer no
advantages when it comes to restoring a checkpoint and incrementally
updating it so that the down time between machines is negligible.

>> My inclination is that create with a specified set of ids is the
>> proper internal kernel API, so we don't have rework things later,
>> because reworking things seems to be a lot more work.
>
> Completely agree with you: the earlier in the object's life we do it, the best
> it is ;-)
>
>> How we want to
>> export this to user space is another matter.
>>
>> One suggestion is to a have /proc or a proc like filesystem that
>> allows us to read, create, and populate files to see all of the
>> application state.
>
> And that would be quite useful for debugging purpose, as you were saying
> earlier.

Yes. Part of why I suggested it in that way. Debugging and
checkpoint/restart have a lot in common. If we can take advantage of
that it would be cool.

Eric

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

---

Subject: Re: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by serue on Thu, 13 Mar 2008 19:06:57 GMT
View Forum Message <> Reply to Message

Quoting Eric W. Biederman (ebiederm@xmission.com):
> Nadia Derbey <Nadia.Derbey@bull.net> writes:
>

> > Eric W. Biederman wrote:
> >> "Nadia Derbey" <Nadia.Derbey@bull.net> writes:
> >>
> >>>A couple of weeks ago, a discussion has started after Pierre's proposal for
> >>>a new syscall to change an ipc id (see thread
> >>>http://lkml.org/lkml/2008/1/29/209).
> >>>
> >>>
> >>>Oren's suggestion was to force an object's id during its creation, rather
> >>>than 1. create it, 2. change its id.
> >>>
> >>>So here is an implementation of what Oren has suggested.
> >>>
> >>>2 new files are defined under /proc/self:
> >>> . next_ipcid --> next id to use for ipc object creation
> >>> . next_pids --> next upid nr(s) to use for next task to be forked
> >>>   (see patch #2 for more details).
> >>>
> >>>When one of these files (or both of them) is filled, a structure pointed to
> >>>by the calling task struct is filled with these ids.
> >>>
> >>>Then, when the object is created, the id(s) present in that structure are
> >>>used, instead of the default ones.
> >>>A couple of weeks ago, a discussion has started after Pierre's proposal for
> >>>a new syscall to change an ipc id (see thread
> >>>http://lkml.org/lkml/2008/1/29/209).
> >>>
> >>>
> >>>Oren's suggestion was to force an object's id during its creation, rather
> >>>than 1. create it, 2. change its id.
> >>>
> >>>So here is an implementation of what Oren has suggested.
> >>>
> >>>2 new files are defined under /proc/self:
> >>> . next_ipcid --> next id to use for ipc object creation
> >>> . next_pids --> next upid nr(s) to use for next task to be forked
> >>>   (see patch #2 for more details).
> >>>
> >>>When one of these files (or both of them) is filled, a structure pointed to
> >>>by the calling task struct is filled with these ids.
> >>>
> >>>Then, when the object is created, the id(s) present in that structure are
> >>>used, instead of the default ones.
> >>
> >>
> >> "Serge E. Hallyn" <serue@us.ibm.com> writes:
> >>
> >>

> >>>Right the alloc_pidmap() changes will probably be pretty much the same
> >>>no matter how we do set_it(), so it's worth discussing. But I'm
> >>>particularly curious to see what opinions are on the sys_setid().
> >>
> >>
> >> A couple of comments. With respect to alloc_pidmap we already have
> >> the necessary controls (a minimum and a maximum) in place for the
> >> allocation. So except for double checking that those controls are exported
> >> in /proc/sys we don't necessarily need to do anything, special.
> >>
> >> Just play games with the minimum pid value before you fork.
> >
> > Excellent idea! It's trus that properly setting things, we can make the loops
> > executed only once in case we want to use a predefined id.
> >
> >>
> >> Second at least to get the memory map correct we need additional
> >> kernel support.
> >>
> >>
> >> Third to actually get the values out it appears we need additional kernel
> >> support as well. From my limited playing with these things at least
> >> parts of the code were easier to implement in the kernel. The hoops
> >> you have to go to restore a single process (without threads) are
> >> absolutely horrendous in user space.
> >
>
> I was thinking in particular of the mm setup above.
>
> > Ok, but if we have a process that belongs to nested namepsaces, we have no other
> > choice than provide its upid nrs hierarchy, right?
>
> At least a part of it. You only have to provide as much of the nested
> pid hiearchy as you are restoring from your checkpoint.
>
> > So, I guess it's more the way it is presented that you don't agree with (of
> > course provided that we are in a user space oriented solution)?
>
> Yes. On the kernel side we are essentially fine.
>
> >> So this patchset whether it is setid or setting the id at creation time
> >> seems to be jumping the gun. For some namespaces renames are valid and
> >> we can support them. For other namespaces setting the id is a big no-no,
> >> and possibly even controlling the id at creation time is a problem (at
> >> least in the general sense).
> >
> > I'm sorry but I'm pretty new in this domain, so I don't see what are the
> > namespaces where setting (or pre-setting) the id would be a problem?

>
> pids to some extent as people use them in all kinds of files. Being
> able to force the pid of another process could make a hard to trigger
> security hole with file permissions absolutely trivial to hit.
>
> Changing the pid on a process would be even worse because then how
> could you send it signals.
>
> I haven't looked close enough to be able to say in situation X it is a
> problem or in situation Y it is clearly not a problem. I just know
> there is a lot that happens with ids and security so we need to tread
> lightly, and carefully.
>
> >> Because if you can easily control the id
> >> you may be able to more easily exploit security holes.
> >>
> >> I'm not at all inclined to make it easy for userspace to rename or set
> >> the id of a new resource unless it already makes sense in that
> >> namespace.
> >>
> >> We need to limit anything in a checkpoint to user space visible
> >> state.
> >
> > OK, but a tasks that belongs to nested pid namespaces is known from other tasks
> > as one of its "intermediate" pids, depending on the namespace level we are
> > considering. So we can consider these "intermediate pids" as user space visible,
> > can't we?
>
> Yes.
>
> > Given the following pid namespaces hierarchy:
> > PNS0 -> PNS1 -> PNS2 -> PNS3 -> PNS4
> > A task that belongs to PSN2 has 3 upid nrs:
> > UP0, UP1, and UP2
> > So:
> > . UP0 can be obtained if we do a "ps -ef" in pid ns #1 (PNS0)
> > . UP1 can be obtained if we do a "ps -ef" in pid ns #1 (PNS1)
> > . UP2 can be obtained if we do a "ps -ef" in pid ns #1 (PNS2)
> >
> > So UP[0-2] are user space visible (again, depending on "where" we are in pid ns
> > hierarchy, aren't they?
>
> Totally.
>
> > Sure, I completely agree with you! So may be the 1st thing to do would be to
> > decide which approach (user space vs kernel) should be adopted for c/r? Sorry if
> > what I'm saying is stupid, but imho a clear answer to this question would make
> > us all go the same direction ==> save time for further investigations.

>
> Agreed, although it isn't quite that cut and dry.
>
> The question is what granularity do we export the checkpoint restore
> functionality with, and do we manage to have it serve additional
> functions as well.
>
> Because ultimately it is user space saying checkpoint and it is user
> space saying restore. Although we need to be careful that there
> are not cheaper migration solutions that we are precluding.
>
> Getting the user space interface correct is going to be the tricky
> and important.
>
>
>
> My inclination is that the cost in migrating a container is the cost
> of moving the data between machines. Which (not counting filesystems)
> would be the anonymous pages and the shared memory areas.

I'm playing around with providing swapfile namespaces. An app expecting to be checkpointed could start itself with unshare(CLONE_NEWSWAP), and would receive an empty swapfile ns. Then it could swapon some nfs mounted file. When it though time for migration might be coming up, it could ask to swap out more of it's unused memory ahead of time to speed up the actual checkpoint. Finally, the checkpoing would be swapping out all the remaining pages, writing the pte info to a file (Dave has mentioned and played with this), and then suspending the rest of the kernel resources.

> So if we have a checkpoint as a directory.
> We could have files for the data of each shared memory region.
>
> We could have files for the data of each shared anonymous region
> shared between mm's.
>
> We could have ELF core files with extra notes to describe the full
> state of each process.
>
> We could have files describing each sysvipc object (shared memory,
> message queues, and semaphores).
>
> Futexes?
>
> I would suggest a directory for each different kind of file, making
> conflicts easier to avoid, and data types easier to predict.
>
>

> Once we have a least one valid checkpoint format the question becomes
> how do we get the checkpoint out of the kernel, and how do we get
> the checkpoint into the kernel. And especially how do we allow for
> incremental data movement so live migration with minimal interruption
> of service is possible.
>
> This means that we need to preload all of the large memory areas into
> shared memory areas or processes. Everything else file descriptors
> and the like I expect will be sufficiently inexpesive that we can
> treat their recreation as instantaneous.
>
>
> If incremental migration of data was not so useful I would say just
> have a single syscall to dump everything, and another syscall to
> restore everything.
>
> Since incremental migration is so useful. My gut feel is
> checkpointfs or something similar where we can repeated read
> and write a checkpoint before we commit to starting it is useful.
>
> I am welcome to other better suggestions from the experience of the
> people who have implemented this before.
>
>
> A very fine grained user space solution where user space createsn
> each object using the traditional APIs and restores them similarly
> seems harder to implement, harder to get right, and harder to
> maintain. In part because it hides what we are actually trying to do.
>
> Further a fine grained user space approach appears to offer no
> advantages when it comes to restoring a checkpoint and incrementally
> updating it so that the down time between machines is negligible.
>
> >> My inclination is that create with a specified set of ids is the
> >> proper internal kernel API, so we don't have rework things later,
> >> because reworking things seems to be a lot more work.
> >
> > Completely agree with you: the earlier in the object's life we do it, the best
> > it is ;-)
> >
> >> How we want to
> >> export this to user space is another matter.
> >>
> >> One suggestion is to a have /proc or a proc like filesystem that
> >> allows us to read, create, and populate files to see all of the
> >> application state.
> >
> > And that would be quite useful for debugging purpose, as you were saying

> > earlier.
>
> Yes. Part of why I suggested it in that way. Debugging and
> checkpoint/restart have a lot in common. If we can take advantage of
> that it would be cool.
>
> Eric

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

---

Subject: Re: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by Oren Laadan on Thu, 13 Mar 2008 20:01:02 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Nadia Derbey <Nadia.Derbey@bull.net> writes:
>
>> Eric W. Biederman wrote:
>>> "Nadia Derbey" <Nadia.Derbey@bull.net> writes:
>>>
>>>> A couple of weeks ago, a discussion has started after Pierre's proposal for
>>>> a new syscall to change an ipc id (see thread
>>>> http://lkml.org/lkml/2008/1/29/209).
>>>>
>>>>
>>>> Oren's suggestion was to force an object's id during its creation, rather
>>>> than 1. create it, 2. change its id.
>>>>
>>>> So here is an implementation of what Oren has suggested.
>>>>
>>>> 2 new files are defined under /proc/self:
>>>> . next_ipcid --> next id to use for ipc object creation
>>>> . next_pids --> next upid nr(s) to use for next task to be forked
>>>>   (see patch #2 for more details).
>>>>
>>>> When one of these files (or both of them) is filled, a structure pointed to
>>>> by the calling task struct is filled with these ids.
>>>>
>>>> Then, when the object is created, the id(s) present in that structure are
>>>> used, instead of the default ones.
>>>> A couple of weeks ago, a discussion has started after Pierre's proposal for
>>>> a new syscall to change an ipc id (see thread
>>>> http://lkml.org/lkml/2008/1/29/209).
>>>>
>>>>

>>>> Oren's suggestion was to force an object's id during its creation, rather
>>>> than 1. create it, 2. change its id.
>>>>
>>>> So here is an implementation of what Oren has suggested.
>>>>
>>>> 2 new files are defined under /proc/self:
>>>> . next_ipcid --> next id to use for ipc object creation
>>>> . next_pids --> next upid nr(s) to use for next task to be forked
>>>> (see patch #2 for more details).
>>>>
>>>> When one of these files (or both of them) is filled, a structure pointed to
>>>> by the calling task struct is filled with these ids.
>>>>
>>>> Then, when the object is created, the id(s) present in that structure are
>>>> used, instead of the default ones.
>>>
>>> "Serge E. Hallyn" <serue@us.ibm.com> writes:
>>>
>>>
>>>> Right the alloc_pidmap() changes will probably be pretty much the same
>>>> no matter how we do set_it(), so it's worth discussing. But I'm
>>>> particularly curious to see what opinions are on the sys_setid().
>>>
>>> A couple of comments. With respect to alloc_pidmap we already have
>>> the necessary controls (a minimum and a maximum) in place for the
>>> allocation. So except for double checking that those controls are exported
>>> in /proc/sys we don't necessarily need to do anything, special.
>>>
>>> Just play games with the minimum pid value before you fork.
>> Excellent idea! It's trus that properly setting things, we can make the loops
>> executed only once in case we want to use a predefined id.
>>
>>> Second at least to get the memory map correct we need additional
>>> kernel support.
>>>
>>>
>>> Third to actually get the values out it appears we need additional kernel
>>> support as well. From my limited playing with these things at least
>>> parts of the code were easier to implement in the kernel. The hoops
>>> you have to go to restore a single process (without threads) are
>>> absolutely horrendous in user space.
>
> I was thinking in particular of the mm setup above.
>
>> Ok, but if we have a process that belongs to nested namepsaces, we have no other
>> choice than provide its upid nrs hierarchy, right?
>
> At least a part of it. You only have to provide as much of the nested

> pid hiearchy as you are restoring from your checkpoint.
>
>> So, I guess it's more the way it is presented that you don't agree with (of
>> course provided that we are in a user space oriented solution)?
>
> Yes. On the kernel side we are essentially fine.
>
>>> So this patchset whether it is setid or setting the id at creation time
>>> seems to be jumping the gun. For some namespaces renames are valid and
>>> we can support them. For other namespaces setting the id is a big no-no,
>>> and possibly even controlling the id at creation time is a problem (at
>>> least in the general sense).
>> I'm sorry but I'm pretty new in this domain, so I don't see what are the
>> namespaces where setting (or pre-setting) the id would be a problem?
>
> pids to some extent as people use them in all kinds of files. Being
> able to force the pid of another process could make a hard to trigger
> security hole with file permissions absolutely trivial to hit.

Since the intent of this mechanism is to allow ckpt/restart, it makes
sense to only allow this operation during restart. For example, in zap,
containers have a state, e.g. running, stopped, ckpt, restart, and this
is only possible in restart state; Furthermore, a container can only be
put in restart state at creation time, and only by root. Of course, you
should only trust that as much as you trust the root :O

>
> Changing the pid on a process would be even worse because then how
> could you send it signals.
>
> I haven't looked close enough to be able to say in situation X it is a
> problem or in situation Y it is clearly not a problem. I just know
> there is a lot that happens with ids and security so we need to tread
> lightly, and carefully.
>
>>> Because if you can easily control the id
>>> you may be able to more easily exploit security holes.
>>>
>>> I'm not at all inclined to make it easy for userspace to rename or set
>>> the id of a new resource unless it already makes sense in that
>>> namespace.
>>>
>>> We need to limit anything in a checkpoint to user space visible
>>> state.
>> OK, but a tasks that belongs to nested pid namespaces is known from other tasks
>> as one of its "intermediate" pids, depending on the namespace level we are
>> considering. So we can consider these "intermediate pids" as user space visible,
>> can't we?

>
> Yes.
>
>> Given the following pid namespaces hierarchy:
>> PNS0 -> PNS1 -> PNS2 -> PNS3 -> PNS4
>> A task that belongs to PSN2 has 3 upid nrs:
>> UP0, UP1, and UP2
>> So:
>> . UP0 can be obtained if we do a "ps -ef" in pid ns #1 (PNS0)
>> . UP1 can be obtained if we do a "ps -ef" in pid ns #1 (PNS1)
>> . UP2 can be obtained if we do a "ps -ef" in pid ns #1 (PNS2)
>>
>> So UP[0-2] are user space visible (again, depending on "where" we are in pid ns
>> hierarchy, aren't they?
>
> Totally.
>
>> Sure, I completely agree with you! So may be the 1st thing to do would be to
>> decide which approach (user space vs kernel) should be adopted for c/r? Sorry if
>> what I'm saying is stupid, but imho a clear answer to this question would make
>> us all go the same direction ==> save time for further investigations.
>
> Agreed, although it isn't quite that cut and dry.
>
> The question is what granularity do we export the checkpoint restore
> functionality with, and do we manage to have it serve additional
> functions as well.
>
> Because ultimately it is user space saying checkpoint and it is user
> space saying restore. Although we need to be careful that there
> are not cheaper migration solutions that we are precluding.
>
> Getting the user space interface correct is going to be the tricky
> and important.
>
>
>
> My inclination is that the cost in migrating a container is the cost
> of moving the data between machines. Which (not counting filesystems)
> would be the anonymous pages and the shared memory areas.
>
> So if we have a checkpoint as a directory.
> We could have files for the data of each shared memory region.
>
> We could have files for the data of each shared anonymous region
> shared between mm's.
>
> We could have ELF core files with extra notes to describe the full

> state of each process.
>
> We could have files describing each sysvipc object (shared memory,
> message queues, and semaphores).
>
> Futexes?
>
> I would suggest a directory for each different kind of file, making
> conflicts easier to avoid, and data types easier to predict.
>
>
> Once we have a least one valid checkpoint format the question becomes
> how do we get the checkpoint out of the kernel, and how do we get
> the checkpoint into the kernel. And especially how do we allow for
> incremental data movement so live migration with minimal interruption
> of service is possible.
>
> This means that we need to preload all of the large memory areas into
> shared memory areas or processes. Everything else file descriptors
> and the like I expect will be sufficiently inexpesive that we can
> treat their recreation as instantaneous.
>
>
> If incremental migration of data was not so useful I would say just
> have a single syscall to dump everything, and another syscall to
> restore everything.
>
> Since incremental migration is so useful. My gut feel is
> checkpointfs or something similar where we can repeated read
> and write a checkpoint before we commit to starting it is useful.
>
> I am welcome to other better suggestions from the experience of the
> people who have implemented this before.
>
>
> A very fine grained user space solution where user space createsn
> each object using the traditional APIs and restores them similarly
> seems harder to implement, harder to get right, and harder to
> maintain. In part because it hides what we are actually trying to do.
>
> Further a fine grained user space approach appears to offer no
> advantages when it comes to restoring a checkpoint and incrementally
> updating it so that the down time between machines is negligible.
>
>>> My inclination is that create with a specified set of ids is the
>>> proper internal kernel API, so we don't have rework things later,
>>> because reworking things seems to be a lot more work.
>> Completely agree with you: the earlier in the object's life we do it, the best

>> it is ;-)
>>
>>> How we want to
>>> export this to user space is another matter.
>>>
>>> One suggestion is to a have /proc or a proc like filesystem that
>>> allows us to read, create, and populate files to see all of the
>>> application state.
>> And that would be quite useful for debugging purpose, as you were saying
>> earlier.
>
> Yes. Part of why I suggested it in that way. Debugging and
> checkpoint/restart have a lot in common. If we can take advantage of
> that it would be cool.
>
> Eric
> _____
> 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

---

Subject: Re: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by ebiederm on Thu, 13 Mar 2008 23:12:40 GMT
View Forum Message <> Reply to Message

Oren Laadan <orenl@cs.columbia.edu> writes:

>>> I'm sorry but I'm pretty new in this domain, so I don't see what are the
>>> namespaces where setting (or pre-setting) the id would be a problem?
>>
>> pids to some extent as people use them in all kinds of files. Being
>> able to force the pid of another process could make a hard to trigger
>> security hole with file permissions absolutely trivial to hit.
>
> Since the intent of this mechanism is to allow ckpt/restart, it makes
> sense to only allow this operation during restart. For example, in zap,
> containers have a state, e.g. running, stopped, ckpt, restart, and this
> is only possible in restart state; Furthermore, a container can only be
> put in restart state at creation time, and only by root. Of course, you
> should only trust that as much as you trust the root :O

Yes and thanks.

The notion of the state of a container makes a lot of sense (even if
we never implement explicit state bits).

Eric

_____
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 specified id
Posted by Oren Laadan on Thu, 13 Mar 2008 23:16:07 GMT
View Forum Message <> Reply to Message

Nadia.Derbey@bull.net wrote:
> A couple of weeks ago, a discussion has started after Pierre's proposal for
> a new syscall to change an ipc id (see thread
> http://lkml.org/lkml/2008/1/29/209).
>
>
> Oren's suggestion was to force an object's id during its creation, rather
> than 1. create it, 2. change its id.
>
> So here is an implementation of what Oren has suggested.
>
> 2 new files are defined under /proc/self:
> . next_ipcid --> next id to use for ipc object creation
> . next_pids --> next upid nr(s) to use for next task to be forked
> (see patch #2 for more details).

Generally looks good. One meta-comment, though:

I wonder why you use separate files for separate resources, and why you'd
want to write multiple identifiers in one go; it seems to complicate the
code and interface with minimal gain.
In practice, a process will only do either one or the other, so a single
file is enough (e.g. "next_id").
Also, writing a single value at a time followed by the syscall is enough;
it's definitely not a performance issue to have multiple calls.
We assume the user/caller knows what she's doing, so no need to classify
the identifier (that is, tell the kernel it's a pid, or an ipc id) ahead
of time. The caller simply writes a value and then calls the relevant
syscall, or otherwise the results may not be what she expected...
If such context is expected to be required (although I don't see any at
the moment), we can require that the user write "TYPE VALUE" pair to
the "next_id" file.

>

> When one of these files (or both of them) is filled, a structure pointed to
> by the calling task struct is filled with these ids.
>
> Then, when the object is created, the id(s) present in that structure are
> used, instead of the default ones.
>
> The patches are against 2.6.25-rc3-mm1, in the following order:
>
> [PATCH 1/4] adds the procfs facility for next ipc to be created.
> [PATCH 2/4] adds the procfs facility for next task to be forked.
> [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.
>
> Cc-ing Pavel and Sukadev, since they are the pid namespace authors.
>
> Regards,
> Nadia
>
> --
>
> --

_____

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

---

Subject: Re: [RFC][PATCH 4/4] PID: use the target ID specified in procfs
Posted by Oren Laadan on Thu, 13 Mar 2008 23:24:10 GMT

View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Oren Laadan <orenl@cs.columbia.edu> writes:
>
>>>> I'm sorry but I'm pretty new in this domain, so I don't see what are the
>>>> namespaces where setting (or pre-setting) the id would be a problem?
>>> pids to some extent as people use them in all kinds of files. Being
>>> able to force the pid of another process could make a hard to trigger
>>> security hole with file permissions absolutely trivial to hit.
>> Since the intent of this mechanism is to allow ckpt/restart, it makes
>> sense to only allow this operation during restart. For example, in zap,
>> containers have a state, e.g. running, stopped, ckpt, restart, and this
>> is only possible in restart state; Furthermore, a container can only be
>> put in restart state at creation time, and only by root. Of course, you

>> should only trust that as much as you trust the root :O
>
> Yes and thanks.
>
> The notion of the state of a container makes a lot of sense (even if
> we never implement explicit state bits).

I found it extremely helpful in managing containers (pods) in zap. There are three more states, actually: stopping, reviving and dead. It is like extending the notion of process state into their collective representation which is the container. In fact, restricting certain operations to specific states was instrumental in eliminating a myriad of potential races in the implementation.

I believe this belongs to the ever-pending ckpt/restart discussion :)

Oren.

>
> Eric

_____
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 specified id
Posted by Nadia Derbey on Fri, 14 Mar 2008 06:21:15 GMT
View Forum Message <> Reply to Message

Oren Laadan wrote:
>
>
> Nadia.Derbey@bull.net wrote:
>
>> A couple of weeks ago, a discussion has started after Pierre's
>> proposal for
>> a new syscall to change an ipc id (see thread
>> http://lkml.org/lkml/2008/1/29/209).
>>
>>
>> Oren's suggestion was to force an object's id during its creation, rather
>> than 1. create it, 2. change its id.
>>
>> So here is an implementation of what Oren has suggested.
>>
>> 2 new files are defined under /proc/self:
>> . next_ipcid --> next id to use for ipc object creation

>>   . next_pids --> next upid nr(s) to use for next task to be forked
>>   (see patch #2 for more details).
>
>
> Generally looks good. One meta-comment, though:
>
> I wonder why you use separate files for separate resources,

That would be needed in a situation wheere we don't care about next,
say, ipc id to be created but we need a predefined pid. But I must admit
I don't see any pratical application to it.

> and why you'd
> want to write multiple identifiers in one go;

I used multiple identifiers only for the pid values: this is because
when a new pid value is allocated for a process that belongs to nested
namespaces, the lower level upid nr values are allocated in a single
shot. (see alloc_pid()).

> it seems to complicate the
> code and interface with minimal gain.
> In practice, a process will only do either one or the other, so a single
> file is enough (e.g. "next_id").
> Also, writing a single value at a time followed by the syscall is enough;
> it's definitely not a performance issue to have multiple calls.
> We assume the user/caller knows what she's doing, so no need to classify
> the identifier (that is, tell the kernel it's a pid, or an ipc id) ahead
> of time. The caller simply writes a value and then calls the relevant
> syscall, or otherwise the results may not be what she expected...
> If such context is expected to be required (although I don't see any at
> the moment), we can require that the user write "TYPE VALUE" pair to
> the "next_id" file.

That's exactly what I wanted to avoid by creating 1 file per object.
Now, it's true that in a restart context where I guess that things will
be done synchronously, we could have a single next_id file.

>
>>
>> When one of these files (or both of them) is filled, a structure
>> pointed to
>> by the calling task struct is filled with these ids.
>>
>> Then, when the object is created, the id(s) present in that structure are
>> used, instead of the default ones.
>>
>> The patches are against 2.6.25-rc3-mm1, in the following order:

>>
>> [PATCH 1/4] adds the procfs facility for next ipc to be created.
>> [PATCH 2/4] adds the procfs facility for next task to be forked.
>> [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.
>>
>> Cc-ing Pavel and Sukadev, since they are the pid namespace authors.
>>
>> Regards,
>> Nadia
>>
>> --
>>
>> --
>
>
>

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 specified id
Posted by Oren Laadan on Fri, 14 Mar 2008 15:50:13 GMT
View Forum Message <> Reply to Message

Nadia Derbey wrote:
> Oren Laadan wrote:
>>
>>
>> Nadia.Derbey@bull.net wrote:
>>
>>> A couple of weeks ago, a discussion has started after Pierre's
>>> proposal for
>>> a new syscall to change an ipc id (see thread
>>> http://lkml.org/lkml/2008/1/29/209).

>>>
>>>
>>> Oren's suggestion was to force an object's id during its creation,
>>> rather
>>> than 1. create it, 2. change its id.
>>>
>>> So here is an implementation of what Oren has suggested.
>>>
>>> 2 new files are defined under /proc/self:
>>>   . next_ipcid --> next id to use for ipc object creation
>>>   . next_pids --> next upid nr(s) to use for next task to be forked
>>>   (see patch #2 for more details).
>>
>>
>> Generally looks good. One meta-comment, though:
>>
>> I wonder why you use separate files for separate resources,
>
> That would be needed in a situation wheere we don't care about next,
> say, ipc id to be created but we need a predefined pid. But I must admit
> I don't see any pratical application to it.

exactly; why set the next-ipc value so far in advance ? I think it's
better (and less confusing) if we require that setting the next-id value
be done right before the respective syscall.

>
>> and why you'd
>> want to write multiple identifiers in one go;
>
> I used multiple identifiers only for the pid values: this is because
> when a new pid value is allocated for a process that belongs to nested
> namespaces, the lower level upid nr values are allocated in a single
> shot. (see alloc_pid()).
>
>> it seems to complicate the
>> code and interface with minimal gain.
>> In practice, a process will only do either one or the other, so a single
>> file is enough (e.g. "next_id").
>> Also, writing a single value at a time followed by the syscall is enough;
>> it's definitely not a performance issue to have multiple calls.
>> We assume the user/caller knows what she's doing, so no need to classify
>> the identifier (that is, tell the kernel it's a pid, or an ipc id) ahead
>> of time. The caller simply writes a value and then calls the relevant
>> syscall, or otherwise the results may not be what she expected...
>> If such context is expected to be required (although I don't see any at
>> the moment), we can require that the user write "TYPE VALUE" pair to
>> the "next_id" file.

>
> That's exactly what I wanted to avoid by creating 1 file per object.
> Now, it's true that in a restart context where I guess that things will
> be done synchronously, we could have a single next_id file.
>
>>
>>>
>>> When one of these files (or both of them) is filled, a structure
>>> pointed to
>>> by the calling task struct is filled with these ids.
>>>
>>> Then, when the object is created, the id(s) present in that structure
>>> are
>>> used, instead of the default ones.
>>>
>>> The patches are against 2.6.25-rc3-mm1, in the following order:
>>>
>>> [PATCH 1/4] adds the procfs facility for next ipc to be created.
>>> [PATCH 2/4] adds the procfs facility for next task to be forked.
>>> [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.
>>>
>>> Cc-ing Pavel and Sukadev, since they are the pid namespace authors.
>>>
>>> Regards,
>>> Nadia
>>>
>>> --
>>>
>>> --
>>
>>
>>
>
>
> 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 specified id
Posted by Pavel Emelianov on Fri, 14 Mar 2008 15:56:49 GMT
View Forum Message <> Reply to Message

Oren Laadan wrote:
>
> Nadia Derbey wrote:
>> Oren Laadan wrote:
>>>
>>> Nadia.Derbey@bull.net wrote:
>>>
>>>> A couple of weeks ago, a discussion has started after Pierre's
>>>> proposal for
>>>> a new syscall to change an ipc id (see thread
>>>> http://lkml.org/lkml/2008/1/29/209).
>>>>
>>>>
>>>> Oren's suggestion was to force an object's id during its creation,
>>>> rather
>>>> than 1. create it, 2. change its id.
>>>>
>>>> So here is an implementation of what Oren has suggested.
>>>>
>>>> 2 new files are defined under /proc/self:
>>>> . next_ipcid --> next id to use for ipc object creation
>>>> . next_pids --> next upid nr(s) to use for next task to be forked
>>>> (see patch #2 for more details).
>>>
>>> Generally looks good. One meta-comment, though:
>>>
>>> I wonder why you use separate files for separate resources,
>> That would be needed in a situation wheere we don't care about next,
>> say, ipc id to be created but we need a predefined pid. But I must admit
>> I don't see any pratical application to it.
>
> exactly; why set the next-ipc value so far in advance ? I think it's
> better (and less confusing) if we require that setting the next-id value
> be done right before the respective syscall.

And race with some other syscall caller? This will only work if the next-ipc-id
and the next-pid are on a task_struct. Are they (at least supposed to be such)?

_____
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 specified id

Posted by Oren Laadan on Fri, 14 Mar 2008 16:02:46 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> Oren Laadan wrote:
>> Nadia Derbey wrote:
>>> Oren Laadan wrote:
>>>> Nadia.Derbey@bull.net wrote:
>>>>
>>>>> A couple of weeks ago, a discussion has started after Pierre's
>>>>> proposal for
>>>>> a new syscall to change an ipc id (see thread
>>>>> http://lkml.org/lkml/2008/1/29/209).
>>>>>
>>>>>
>>>>> Oren's suggestion was to force an object's id during its creation,
>>>>> rather
>>>>> than 1. create it, 2. change its id.
>>>>>
>>>>> So here is an implementation of what Oren has suggested.
>>>>>
>>>>> 2 new files are defined under /proc/self:
>>>>> . next_ipcid --> next id to use for ipc object creation
>>>>> . next_pids --> next upid nr(s) to use for next task to be forked
>>>>> (see patch #2 for more details).
>>>> Generally looks good. One meta-comment, though:
>>>>
>>>> I wonder why you use separate files for separate resources,
>>> That would be needed in a situation wheere we don't care about next,
>>> say, ipc id to be created but we need a predefined pid. But I must admit
>>> I don't see any pratical application to it.
>> exactly; why set the next-ipc value so far in advance ? I think it's
>> better (and less confusing) if we require that setting the next-id value
>> be done right before the respective syscall.
>
> And race with some other syscall caller? This will only work if the next-ipc-id
> and the next-pid are on a task_struct. Are they (at least supposed to be such)?

yes. that's the first detail I looked for in the patch :)

_____
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 specified id
Posted by Pavel Emelianov on Fri, 14 Mar 2008 16:08:38 GMT

Oren Laadan wrote:
>
> Pavel Emelyanov wrote:
>> Oren Laadan wrote:
>>> Nadia Derbey wrote:
>>>> Oren Laadan wrote:
>>>>> Nadia.Derbey@bull.net wrote:
>>>>>
>>>>>> A couple of weeks ago, a discussion has started after Pierre's
>>>>>> proposal for
>>>>>> a new syscall to change an ipc id (see thread
>>>>>> http://lkml.org/lkml/2008/1/29/209).
>>>>>>
>>>>>>
>>>>>> Oren's suggestion was to force an object's id during its creation,
>>>>>> rather
>>>>>> than 1. create it, 2. change its id.
>>>>>>
>>>>>> So here is an implementation of what Oren has suggested.
>>>>>>
>>>>>> 2 new files are defined under /proc/self:
>>>>>> . next_ipcid --> next id to use for ipc object creation
>>>>>> . next_pids --> next upid nr(s) to use for next task to be forked
>>>>>> (see patch #2 for more details).
>>>>> Generally looks good. One meta-comment, though:
>>>>>
>>>>> I wonder why you use separate files for separate resources,
>>>> That would be needed in a situation wheere we don't care about next,
>>>> say, ipc id to be created but we need a predefined pid. But I must admit
>>>> I don't see any pratical application to it.
>>> exactly; why set the next-ipc value so far in advance ? I think it's
>>> better (and less confusing) if we require that setting the next-id value
>>> be done right before the respective syscall.
>> And race with some other syscall caller? This will only work if the next-ipc-id
>> and the next-pid are on a task_struct. Are they (at least supposed to be such)?
>
> yes. that's the first detail I looked for in the patch :)

OK :) I just remembered some talks about using last_pid for pid allocations
and just wanted to be sure.

_____
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 specified id
Posted by Nadia Derbey on Fri, 14 Mar 2008 16:11:22 GMT
View Forum Message <> Reply to Message

Oren Laadan wrote:
>
>
> Nadia Derbey wrote:
>
>> Oren Laadan wrote:
>>
>>>
>>>
>>> Nadia.Derbey@bull.net wrote:
>>>
>>>> A couple of weeks ago, a discussion has started after Pierre's
>>>> proposal for
>>>> a new syscall to change an ipc id (see thread
>>>> http://lkml.org/lkml/2008/1/29/209).
>>>>
>>>>
>>>> Oren's suggestion was to force an object's id during its creation,
>>>> rather
>>>> than 1. create it, 2. change its id.
>>>>
>>>> So here is an implementation of what Oren has suggested.
>>>>
>>>> 2 new files are defined under /proc/self:
>>>> . next_ipcid --> next id to use for ipc object creation
>>>> . next_pids --> next upid nr(s) to use for next task to be forked
>>>> (see patch #2 for more details).
>>>
>>>
>>>
>>> Generally looks good. One meta-comment, though:
>>>
>>> I wonder why you use separate files for separate resources,
>>
>>
>> That would be needed in a situation wheere we don't care about next,
>> say, ipc id to be created but we need a predefined pid. But I must
>> admit I don't see any pratical application to it.
>
>
> exactly; why set the next-ipc value so far in advance ? I think it's
> better (and less confusing) if we require that setting the next-id value
> be done right before the respective syscall.

Ok, but this "requirement" should be widely agreed upon ;-)

What I mean here is that the solution with 1 file per "object type" can
easily be extended imho:
I don't know how the restart is supposed to work, but we can imagine
feeding all these files with all the object ids just before restart and
let the process pick up the objects ids as it needs them.
Of course, this would require to enhance the files formats, as well as
the way things are stored in the task_struct.

Hope what I'm saying is not too stupid ;-) ?

Regards,
Nadia

>
>>
>>> and why you'd
>>> want to write multiple identifiers in one go;
>>
>>
>> I used multiple identifiers only for the pid values: this is because
>> when a new pid value is allocated for a process that belongs to nested
>> namespaces, the lower level upid nr values are allocated in a single
>> shot. (see alloc_pid()).
>>
>>> it seems to complicate the
>>> code and interface with minimal gain.
>>> In practice, a process will only do either one or the other, so a single
>>> file is enough (e.g. "next_id").
>>> Also, writing a single value at a time followed by the syscall is
>>> enough;
>>> it's definitely not a performance issue to have multiple calls.
>>> We assume the user/caller knows what she's doing, so no need to classify
>>> the identifier (that is, tell the kernel it's a pid, or an ipc id) ahead
>>> of time. The caller simply writes a value and then calls the relevant
>>> syscall, or otherwise the results may not be what she expected...
>>> If such context is expected to be required (although I don't see any at
>>> the moment), we can require that the user write "TYPE VALUE" pair to
>>> the "next_id" file.
>>
>>
>> That's exactly what I wanted to avoid by creating 1 file per object.
>> Now, it's true that in a restart context where I guess that things
>> will be done synchronously, we could have a single next_id file.
>>
>>>
>>>>
>>>> When one of these files (or both of them) is filled, a structure

>>>> pointed to
>>>> by the calling task struct is filled with these ids.
>>>>
>>>> Then, when the object is created, the id(s) present in that
>>>> structure are
>>>> used, instead of the default ones.
>>>>
>>>> The patches are against 2.6.25-rc3-mm1, in the following order:
>>>>
>>>> [PATCH 1/4] adds the procfs facility for next ipc to be created.
>>>> [PATCH 2/4] adds the procfs facility for next task to be forked.
>>>> [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.
>>>>
>>>> Cc-ing Pavel and Sukadev, since they are the pid namespace authors.
>>>>
>>>> Regards,
>>>> Nadia
>>>>
>>>> --
>>>>
>>>> --
>>>
>>>
>>>
>>>
>>
>>
>> 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 specified id
Posted by Nadia Derbey on Fri, 14 Mar 2008 16:11:34 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> Oren Laadan wrote:
>
>>Nadia Derbey wrote:
>>
>>>Oren Laadan wrote:
>>>
>>>>Nadia.Derbey@bull.net wrote:
>>>>
>>>>
>>>>>A couple of weeks ago, a discussion has started after Pierre's
>>>>>proposal for
>>>>>a new syscall to change an ipc id (see thread
>>>>>http://lkml.org/lkml/2008/1/29/209).
>>>>>
>>>>>
>>>>>Oren's suggestion was to force an object's id during its creation,
>>>>>rather
>>>>>than 1. create it, 2. change its id.
>>>>>
>>>>>So here is an implementation of what Oren has suggested.
>>>>>
>>>>>2 new files are defined under /proc/self:
>>>>> . next_ipcid --> next id to use for ipc object creation
>>>>> . next_pids --> next upid nr(s) to use for next task to be forked
>>>>>   (see patch #2 for more details).
>>>>
>>>>Generally looks good. One meta-comment, though:
>>>>
>>>>I wonder why you use separate files for separate resources,
>>>
>>>That would be needed in a situation wheere we don't care about next,
>>>say, ipc id to be created but we need a predefined pid. But I must admit
>>>I don't see any pratical application to it.
>>
>>exactly; why set the next-ipc value so far in advance ? I think it's
>>better (and less confusing) if we require that setting the next-id value
>>be done right before the respective syscall.
>
>
> And race with some other syscall caller? This will only work if the next-ipc-id
> and the next-pid are on a task_struct. Are they (at least supposed to be such)?
>
>

Yes they are.

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 specified id
Posted by Oren Laadan on Fri, 14 Mar 2008 16:45:31 GMT
View Forum Message <> Reply to Message

Nadia Derbey wrote:
> Oren Laadan wrote:
>>
>>
>> Nadia Derbey wrote:
>>
>>> Oren Laadan wrote:
>>>
>>>>
>>>>
>>>> Nadia.Derbey@bull.net wrote:
>>>>
>>>>> A couple of weeks ago, a discussion has started after Pierre's
>>>>> proposal for
>>>>> a new syscall to change an ipc id (see thread
>>>>> http://lkml.org/lkml/2008/1/29/209).
>>>>>
>>>>>
>>>>> Oren's suggestion was to force an object's id during its creation,
>>>>> rather
>>>>> than 1. create it, 2. change its id.
>>>>>
>>>>> So here is an implementation of what Oren has suggested.
>>>>>
>>>>> 2 new files are defined under /proc/self:
>>>>>   . next_ipcid --> next id to use for ipc object creation
>>>>>   . next_pids --> next upid nr(s) to use for next task to be forked
>>>>>   (see patch #2 for more details).
>>>>
>>>>
>>>>
>>>> Generally looks good. One meta-comment, though:
>>>>
>>>> I wonder why you use separate files for separate resources,

>>>
>>>
>>> That would be needed in a situation wheere we don't care about next,
>>> say, ipc id to be created but we need a predefined pid. But I must
>>> admit I don't see any pratical application to it.
>>
>>
>> exactly; why set the next-ipc value so far in advance ? I think it's
>> better (and less confusing) if we require that setting the next-id value
>> be done right before the respective syscall.
>
> Ok, but this "requirement" should be widely agreed upon ;-)

A discussion on the overall checkpoint/restart policy is certainly due
(and increasingly noted recently).

> What I mean here is that the solution with 1 file per "object type" can
> easily be extended imho:

I'm aiming at simplicity and minimal (but not restrictive) API for user
space. I argue that we never really need more than one predetermined value
at a time (eg see below), and the cost of setting such value is so small
that there is no real benefit in setting more than one at a time (either
via multiple files or via an array of values). If in fact you wanted more
than one type at a time, you could still make it happen with a single
file without adding many user-visible files in /proc/<pid>.

So far, I can't think of any such identifier that we'd like to pre-set
that does not fit into a "long" type; simply because the kernel does not
use such identifiers in the first place (pid, ipc, pty#, vc# .. etc). To
be on the safe side, we can require that the format be "long VAL", just
in case (and later you could have other formats).

The only exception, perhaps, is if a TCP connection is rebuilt with a,
say, connect() syscall, and some information needs to be "predetermined"
so we'll need to extend the format. That can be done with another type
eg. "tcp ....." or a separate file (per your view), _then_, not now.
(As a side note, I don't suggest that this is how TCP will be restored).

In any event, the bottom line is that a single file, with a single
value at a time (possibly annotated with a type), is the simplest, and
isn't restrictive, for our purposes. Looking one step ahead, simplicity
and minimal commitment to user space is important in trying to push this
to the mainline kernel...

> I don't know how the restart is supposed to work, but we can imagine
> feeding all these files with all the object ids just before restart and

Building on my own experience with zap I envision the restart operation
of a given task occurring in the context of that task. (I assume this is
how restart will work). Therefore, it makes much sense that before every
syscall that requires a pre-determined resource identifier (eg. clone,
ipc, pty allocation), the task will place the desired value in "next_id"
(and that will only be meaningful during restart) and invoke the said
syscall. Voila.

Note that the restart will "rebuild" the container's state (and the task
state) as it reads in the data from some source. It is likely that not
all data will be available when the first said syscall is about to be
invoked, so you may not be able to feed everything ahead of time.

> let the process pick up the objects ids as it needs them.
> Of course, this would require to enhance the files formats, as well as
> the way things are stored in the task_struct.
>
> Hope what I'm saying is not too stupid ;-) ?
>
> Regards,
> Nadia
>
>>
>>>
>>>> and why you'd
>>>> want to write multiple identifiers in one go;
>>>
>>>
>>> I used multiple identifiers only for the pid values: this is because
>>> when a new pid value is allocated for a process that belongs to
>>> nested namespaces, the lower level upid nr values are allocated in a
>>> single shot. (see alloc_pid()).
>>>
>>>> it seems to complicate the
>>>> code and interface with minimal gain.
>>>> In practice, a process will only do either one or the other, so a
>>>> single
>>>> file is enough (e.g. "next_id").
>>>> Also, writing a single value at a time followed by the syscall is
>>>> enough;
>>>> it's definitely not a performance issue to have multiple calls.
>>>> We assume the user/caller knows what she's doing, so no need to
>>>> classify
>>>> the identifier (that is, tell the kernel it's a pid, or an ipc id)
>>>> ahead
>>>> of time. The caller simply writes a value and then calls the relevant
>>>> syscall, or otherwise the results may not be what she expected...

>>>> If such context is expected to be required (although I don't see any at
>>>> the moment), we can require that the user write "TYPE VALUE" pair to
>>>> the "next_id" file.
>>>
>>>
>>> That's exactly what I wanted to avoid by creating 1 file per object.
>>> Now, it's true that in a restart context where I guess that things
>>> will be done synchronously, we could have a single next_id file.
>>>
>>>>
>>>>>
>>>>> When one of these files (or both of them) is filled, a structure
>>>>> pointed to
>>>>> by the calling task struct is filled with these ids.
>>>>>
>>>>> Then, when the object is created, the id(s) present in that
>>>>> structure are
>>>>> used, instead of the default ones.
>>>>>
>>>>> The patches are against 2.6.25-rc3-mm1, in the following order:
>>>>>
>>>>> [PATCH 1/4] adds the procfs facility for next ipc to be created.
>>>>> [PATCH 2/4] adds the procfs facility for next task to be forked.
>>>>> [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.
>>>>>
>>>>> Cc-ing Pavel and Sukadev, since they are the pid namespace authors.
>>>>>
>>>>> Regards,
>>>>> Nadia
>>>>>
>>>>> --
>>>>>
>>>>> --
>>>>
>>>>
>>>>
>>>>
>>>
>>>
>>> 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 specified id
Posted by serue on Sun, 16 Mar 2008 03:43:20 GMT
View Forum Message <> Reply to Message

Quoting Oren Laadan (orenl@cs.columbia.edu):
>
>
> Nadia Derbey wrote:
> > Oren Laadan wrote:
> >>
> >>
> >> Nadia Derbey wrote:
> >>
> >>> Oren Laadan wrote:
> >>>
> >>>>
> >>>>
> >>>> Nadia.Derbey@bull.net wrote:
> >>>>
> >>>>> A couple of weeks ago, a discussion has started after Pierre's
> >>>>> proposal for
> >>>>> a new syscall to change an ipc id (see thread
> >>>>> http://lkml.org/lkml/2008/1/29/209).
> >>>>>
> >>>>>
> >>>>> Oren's suggestion was to force an object's id during its creation,
> >>>>> rather
> >>>>> than 1. create it, 2. change its id.
> >>>>>
> >>>>> So here is an implementation of what Oren has suggested.
> >>>>>
> >>>>> 2 new files are defined under /proc/self:
> >>>>> . next_ipcid --> next id to use for ipc object creation
> >>>>> . next_pids --> next upid nr(s) to use for next task to be forked
> >>>>> (see patch #2 for more details).
> >>>>

> >>>>
> >>>>
> >>>> Generally looks good. One meta-comment, though:
> >>>>
> >>>> I wonder why you use separate files for separate resources,
> >>>
> >>>
> >>> That would be needed in a situation wheere we don't care about next,
> >>> say, ipc id to be created but we need a predefined pid. But I must
> >>> admit I don't see any pratical application to it.
> >>
> >>
> >> exactly; why set the next-ipc value so far in advance ? I think it's
> >> better (and less confusing) if we require that setting the next-id value
> >> be done right before the respective syscall.
> >
> > Ok, but this "requirement" should be widely agreed upon ;-)
>
> A discussion on the overall checkpoint/restart policy is certainly due
> (and increasingly noted recently).
>
> > What I mean here is that the solution with 1 file per "object type" can
> > easily be extended imho:
>
> I'm aiming at simplicity and minimal (but not restrictive) API for user
> space. I argue that we never really need more than one predetermined value
> at a time (eg see below), and the cost of setting such value is so small
> that there is no real benefit in setting more than one at a time (either
> via multiple files or via an array of values). If in fact you wanted more
> than one type at a time, you could still make it happen with a single
> file without adding many user-visible files in /proc/<pid>.
>
> So far, I can't think of any such identifier that we'd like to pre-set
> that does not fit into a "long" type;

As Nadia has mentioned, if we have checkpointed a container which has
another pid namespace underneath itself, then we will need to restart
some tasks with two predetermined pids. So we'll need two (or more)
longs for the tasks in deeper namespaces.

> simply because the kernel does not
> use such identifiers in the first place (pid, ipc, pty#, vc# .. etc). To
> be on the safe side, we can require that the format be "long VAL", just
> in case (and later you could have other formats).
>
> The only exception, perhaps, is if a TCP connection is rebuilt with a,
> say, connect() syscall, and some information needs to be "predetermined"
> so we'll need to extend the format. That can be done with another type

> eg. "tcp ....." or a separate file (per your view), _then_, not now.
> (As a side note, I don't suggest that this is how TCP will be restored).
>
> In any event, the bottom line is that a single file, with a single
> value at a time (possibly annotated with a type), is the simplest, and
> isn't restrictive, for our purposes. Looking one step ahead, simplicity
> and minimal commitment to user space is important in trying to push this
> to the mainline kernel...
>
> > I don't know how the restart is supposed to work, but we can imagine
> > feeding all these files with all the object ids just before restart and
>
> Building on my own experience with zap I envision the restart operation
> of a given task occurring in the context of that task.

Could be, but not necessarily the case. Eric has mentioned using elf
files for restart, and that's one way to go, but whether one central
restart task sets up all the children or the children set themselves up
is yet another design point we haven't decided. I would think that
with a centralized restart it would be easier to assure for instance
that shared anon pages would be properly set up and shared, but since
you advocate each-task-starts-itself I trust zap must handle that.

> (I assume this is
> how restart will work). Therefore, it makes much sense that before every
> syscall that requires a pre-determined resource identifier (eg. clone,
> ipc, pty allocation), the task will place the desired value in "next_id"
> (and that will only be meaningful during restart) and invoke the said
> syscall. Voila.
>
> Note that the restart will "rebuild" the container's state (and the task
> state) as it reads in the data from some source. It is likely that not
> all data will be available when the first said syscall is about to be
> invoked, so you may not be able to feed everything ahead of time.
>
>
> > let the process pick up the objects ids as it needs them.
> > Of course, this would require to enhance the files formats, as well as
> > the way things are stored in the task_struct.
> >
> > Hope what I'm saying is not too stupid ;-) ?
> >
> > Regards,
> > Nadia
> >
> >>
> >>>
> >>>> and why you'd

> >>>> want to write multiple identifiers in one go;
> >>>
> >>>
> >>> I used multiple identifiers only for the pid values: this is because
> >>> when a new pid value is allocated for a process that belongs to
> >>> nested namespaces, the lower level upid nr values are allocated in a
> >>> single shot. (see alloc_pid()).
> >>>
> >>>> it seems to complicate the
> >>>> code and interface with minimal gain.
> >>>> In practice, a process will only do either one or the other, so a
> >>>> single
> >>>> file is enough (e.g. "next_id").
> >>>> Also, writing a single value at a time followed by the syscall is
> >>>> enough;
> >>>> it's definitely not a performance issue to have multiple calls.
> >>>> We assume the user/caller knows what she's doing, so no need to
> >>>> classify
> >>>> the identifier (that is, tell the kernel it's a pid, or an ipc id)
> >>>> ahead
> >>>> of time. The caller simply writes a value and then calls the relevant
> >>>> syscall, or otherwise the results may not be what she expected...
> >>>> If such context is expected to be required (although I don't see any at
> >>>> the moment), we can require that the user write "TYPE VALUE" pair to
> >>>> the "next_id" file.
> >>>
> >>>
> >>> That's exactly what I wanted to avoid by creating 1 file per object.
> >>> Now, it's true that in a restart context where I guess that things
> >>> will be done synchronously, we could have a single next_id file.
> >>>
> >>>>
> >>>>>
> >>>>> When one of these files (or both of them) is filled, a structure
> >>>>> pointed to
> >>>>> by the calling task struct is filled with these ids.
> >>>>>
> >>>>> Then, when the object is created, the id(s) present in that
> >>>>> structure are
> >>>>> used, instead of the default ones.
> >>>>>
> >>>>> The patches are against 2.6.25-rc3-mm1, in the following order:
> >>>>>
> >>>>> [PATCH 1/4] adds the procfs facility for next ipc to be created.
> >>>>> [PATCH 2/4] adds the procfs facility for next task to be forked.
> >>>>> [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.
> >>>>>
> >>>>> Cc-ing Pavel and Sukadev, since they are the pid namespace authors.
> >>>>>
> >>>>> Regards,
> >>>>> Nadia
> >>>>>
> >>>>> --
> >>>>>
> >>>>> --
> >>>>
> >>>>
> >>>>
> >>>>
> >>>
> >>>
> >>> 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

---

Subject: Re: [RFC][PATCH 0/4] Object creation with a specified id
Posted by Oren Laadan on Sun, 16 Mar 2008 19:08:44 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Oren Laadan (orenl@cs.columbia.edu):
>>
>> Nadia Derbey wrote:
>>> Oren Laadan wrote:
>>>>

>>>> Nadia Derbey wrote:
>>>>
>>>>> Oren Laadan wrote:
>>>>>
>>>>>>
>>>>>> Nadia.Derbey@bull.net wrote:
>>>>>>
>>>>>>> A couple of weeks ago, a discussion has started after Pierre's
>>>>>>> proposal for
>>>>>>> a new syscall to change an ipc id (see thread
>>>>>>> http://lkml.org/lkml/2008/1/29/209).
>>>>>>>
>>>>>>>
>>>>>>> Oren's suggestion was to force an object's id during its creation,
>>>>>>> rather
>>>>>>> than 1. create it, 2. change its id.
>>>>>>>
>>>>>>> So here is an implementation of what Oren has suggested.
>>>>>>>
>>>>>>> 2 new files are defined under /proc/self:
>>>>>>> . next_ipcid --> next id to use for ipc object creation
>>>>>>> . next_pids --> next upid nr(s) to use for next task to be forked
>>>>>>> (see patch #2 for more details).
>>>>>>
>>>>>>
>>>>>> Generally looks good. One meta-comment, though:
>>>>>>
>>>>>> I wonder why you use separate files for separate resources,
>>>>>
>>>>> That would be needed in a situation wheere we don't care about next,
>>>>> say, ipc id to be created but we need a predefined pid. But I must
>>>>> admit I don't see any pratical application to it.
>>>>
>>>> exactly; why set the next-ipc value so far in advance ? I think it's
>>>> better (and less confusing) if we require that setting the next-id value
>>>> be done right before the respective syscall.
>>> Ok, but this "requirement" should be widely agreed upon ;-)
>> A discussion on the overall checkpoint/restart policy is certainly due
>> (and increasingly noted recently).
>>
>>> What I mean here is that the solution with 1 file per "object type" can
>>> easily be extended imho:
>> I'm aiming at simplicity and minimal (but not restrictive) API for user
>> space. I argue that we never really need more than one predetermined value
>> at a time (eg see below), and the cost of setting such value is so small
>> that there is no real benefit in setting more than one at a time (either
>> via multiple files or via an array of values). If in fact you wanted more
>> than one type at a time, you could still make it happen with a single

>> file without adding many user-visible files in /proc/<pid>.
>>
>> So far, I can't think of any such identifier that we'd like to pre-set
>> that does not fit into a "long" type;
>
> As Nadia has mentioned, if we have checkpointed a container which has
> another pid namespace underneath itself, then we will need to restart
> some tasks with two predetermined pids. So we'll need two (or more)
> longs for the tasks in deeper namespaces.

I see. So more than a single "long" type is probably needed. I'd still
prefer that the "scope" of a preset identifier through "next_id" should
be the subsequent syscall; so if you need multiple values for the next
syscall you use it, but you don't support leftovers for the next syscall
to use. The typing system can be something like "long VAL" and then for
array "long* VAL VAL VAL ...", for instance.

>
>> simply because the kernel does not
>> use such identifiers in the first place (pid, ipc, pty#, vc# .. etc). To
>> be on the safe side, we can require that the format be "long VAL", just
>> in case (and later you could have other formats).
>>
>> The only exception, perhaps, is if a TCP connection is rebuilt with a,
>> say, connect() syscall, and some information needs to be "predetermined"
>> so we'll need to extend the format. That can be done with another type
>> eg. "tcp ....." or a separate file (per your view), _then_, not now.
>> (As a side note, I don't suggest that this is how TCP will be restored).
>>
>> In any event, the bottom line is that a single file, with a single
>> value at a time (possibly annotated with a type), is the simplest, and
>> isn't restrictive, for our purposes. Looking one step ahead, simplicity
>> and minimal commitment to user space is important in trying to push this
>> to the mainline kernel...
>>
>>> I don't know how the restart is supposed to work, but we can imagine
>>> feeding all these files with all the object ids just before restart and
>> Building on my own experience with zap I envision the restart operation
>> of a given task occurring in the context of that task.
>
> Could be, but not necessarily the case. Eric has mentioned using elf
> files for restart, and that's one way to go, but whether one central

I'm not familiar with the details of this.

> restart task sets up all the children or the children set themselves up
> is yet another design point we haven't decided. I would think that
> with a centralized restart it would be easier to assure for instance

> that shared anon pages would be properly set up and shared, but since
> you advocate each-task-starts-itself I trust zap must handle that.

The main reason I think a task should setup itself, is because most of
the setup requires that new resources be allocated, and the kernel is
already centered around this approach that a task allocates for itself,
not for another task. For instance, if you need to restore a VMA, you
simply call mmap(), a new file, you call open() etc.

Shared anon pages are one example of shared resources that may be used
by multiple processes. Zap's approach is to have the "first" user (in
the sense of the first time the resource is seen during checkpoint) do
the actual restore, and place it in a global table, and then subsequent
tasks will find it in the table and "map" it into their view.

Decentralizing also allow multiple tasks to restart concurrently.

Are we ready to start concrete discussion on the architecture for the
checkpoint/restart ? (and if so .. time to change the subject line).

>
>> (I assume this is
>> how restart will work). Therefore, it makes much sense that before every
>> syscall that requires a pre-determined resource identifier (eg. clone,
>> ipc, pty allocation), the task will place the desired value in "next_id"
>> (and that will only be meaningful during restart) and invoke the said
>> syscall. Voila.
>>
>> Note that the restart will "rebuild" the container's state (and the task
>> state) as it reads in the data from some source. It is likely that not
>> all data will be available when the first said syscall is about to be
>> invoked, so you may not be able to feed everything ahead of time.
>>
>>
>>> let the process pick up the objects ids as it needs them.
>>> Of course, this would require to enhance the files formats, as well as
>>> the way things are stored in the task_struct.
>>>
>>> Hope what I'm saying is not too stupid ;-) ?
>>>
>>> Regards,
>>> Nadia
>>>
>>>>>>> and why you'd
>>>>>>> want to write multiple identifiers in one go;
>>>>>
>>>>> I used multiple identifiers only for the pid values: this is because
>>>>> when a new pid value is allocated for a process that belongs to

>>>>> nested namespaces, the lower level upid nr values are allocated in a
>>>>> single shot. (see alloc_pid()).
>>>>>
>>>>>> it seems to complicate the
>>>>>> code and interface with minimal gain.
>>>>>> In practice, a process will only do either one or the other, so a
>>>>>> single
>>>>>> file is enough (e.g. "next_id").
>>>>>> Also, writing a single value at a time followed by the syscall is
>>>>>> enough;
>>>>>> it's definitely not a performance issue to have multiple calls.
>>>>>> We assume the user/caller knows what she's doing, so no need to
>>>>>> classify
>>>>>> the identifier (that is, tell the kernel it's a pid, or an ipc id)
>>>>>> ahead
>>>>>> of time. The caller simply writes a value and then calls the relevant
>>>>>> syscall, or otherwise the results may not be what she expected...
>>>>>> If such context is expected to be required (although I don't see any at
>>>>>> the moment), we can require that the user write "TYPE VALUE" pair to
>>>>>> the "next_id" file.
>>>>>
>>>>> That's exactly what I wanted to avoid by creating 1 file per object.
>>>>> Now, it's true that in a restart context where I guess that things
>>>>> will be done synchronously, we could have a single next_id file.
>>>>>
>>>>>>> When one of these files (or both of them) is filled, a structure
>>>>>>> pointed to
>>>>>>> by the calling task struct is filled with these ids.
>>>>>>>
>>>>>>> Then, when the object is created, the id(s) present in that
>>>>>>> structure are
>>>>>>> used, instead of the default ones.
>>>>>>>
>>>>>>> The patches are against 2.6.25-rc3-mm1, in the following order:
>>>>>>>
>>>>>>> [PATCH 1/4] adds the procfs facility for next ipc to be created.
>>>>>>> [PATCH 2/4] adds the procfs facility for next task to be forked.
>>>>>>> [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.
>>>>>>>
>>>>>>> Cc-ing Pavel and Sukadev, since they are the pid namespace authors.

>>>>>>>
>>>>>>> Regards,
>>>>>>> Nadia
>>>>>>>
>>>>>>> --
>>>>>>>
>>>>>>> --
>>>>>>
>>>>>>
>>>>>>
>>>>>
>>>>> 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

---

Subject: Re: [RFC][PATCH 0/4] Object creation with a specified id
Posted by serue on Mon, 17 Mar 2008 14:44:52 GMT
View Forum Message <> Reply to Message

Quoting Oren Laadan (orenl@cs.columbia.edu):
>
>
> Serge E. Hallyn wrote:
>> Quoting Oren Laadan (orenl@cs.columbia.edu):
>>>
>>> Nadia Derbey wrote:
>>>> Oren Laadan wrote:
>>>>>
>>>>> Nadia Derbey wrote:
>>>>>
>>>>>> Oren Laadan wrote:
>>>>>>>
>>>>>>>
>>>>>>> Nadia.Derbey@bull.net wrote:
>>>>>>>
>>>>>>>> A couple of weeks ago, a discussion has started after Pierre's
>>>>>>>> proposal for

>>>>>>>> a new syscall to change an ipc id (see thread
>>>>>>>> http://lkml.org/lkml/2008/1/29/209).
>>>>>>>>
>>>>>>>>
>>>>>>>> Oren's suggestion was to force an object's id during its creation,
>>>>>>>> rather
>>>>>>>> than 1. create it, 2. change its id.
>>>>>>>>
>>>>>>>> So here is an implementation of what Oren has suggested.
>>>>>>>>
>>>>>>>> 2 new files are defined under /proc/self:
>>>>>>>> . next_ipcid --> next id to use for ipc object creation
>>>>>>>> . next_pids --> next upid nr(s) to use for next task to be forked
>>>>>>>> (see patch #2 for more details).
>>>>>>>>
>>>>>>>>
>>>>>>>> Generally looks good. One meta-comment, though:
>>>>>>>>
>>>>>>>> I wonder why you use separate files for separate resources,
>>>>>>>
>>>>>>> That would be needed in a situation wheere we don't care about next,
>>>>>>> say, ipc id to be created but we need a predefined pid. But I must
>>>>>>> admit I don't see any pratical application to it.
>>>>>>
>>>>>> exactly; why set the next-ipc value so far in advance ? I think it's
>>>>>> better (and less confusing) if we require that setting the next-id
>>>>>> value
>>>>>> be done right before the respective syscall.
>>>> Ok, but this "requirement" should be widely agreed upon ;-)
>>> A discussion on the overall checkpoint/restart policy is certainly due
>>> (and increasingly noted recently).
>>>
>>> What I mean here is that the solution with 1 file per "object type" can
>>> easily be extended imho:
>>> I'm aiming at simplicity and minimal (but not restrictive) API for user
>>> space. I argue that we never really need more than one predetermined
>>> value
>>> at a time (eg see below), and the cost of setting such value is so small
>>> that there is no real benefit in setting more than one at a time (either
>>> via multiple files or via an array of values). If in fact you wanted more
>>> than one type at a time, you could still make it happen with a single
>>> file without adding many user-visible files in /proc/<pid>.
>>>
>>> So far, I can't think of any such identifier that we'd like to pre-set
>>> that does not fit into a "long" type;
>> As Nadia has mentioned, if we have checkpointed a container which has
>> another pid namespace underneath itself, then we will need to restart
>> some tasks with two predetermined pids. So we'll need two (or more)

>> longs for the tasks in deeper namespaces.
>
> I see. So more than a single "long" type is probably needed. I'd still
> prefer that the "scope" of a preset identifier through "next_id" should
> be the subsequent syscall;

> so if you need multiple values for the next
> syscall you use it, but you don't support leftovers for the next syscall
> to use.

Agreed.

> The typing system can be something like "long VAL" and then for
> array "long* VAL VAL VAL ...", for instance.
>
>>> simply because the kernel does not
>>> use such identifiers in the first place (pid, ipc, pty#, vc# .. etc). To
>>> be on the safe side, we can require that the format be "long VAL", just
>>> in case (and later you could have other formats).
>>>
>>> The only exception, perhaps, is if a TCP connection is rebuilt with a,
>>> say, connect() syscall, and some information needs to be "predetermined"
>>> so we'll need to extend the format. That can be done with another type
>>> eg. "tcp ....." or a separate file (per your view), _then_, not now.
>>> (As a side note, I don't suggest that this is how TCP will be restored).
>>>
>>> In any event, the bottom line is that a single file, with a single
>>> value at a time (possibly annotated with a type), is the simplest, and
>>> isn't restrictive, for our purposes. Looking one step ahead, simplicity
>>> and minimal commitment to user space is important in trying to push this
>>> to the mainline kernel...
>>>
>>>> I don't know how the restart is supposed to work, but we can imagine
>>>> feeding all these files with all the object ids just before restart and
>>> Building on my own experience with zap I envision the restart operation
>>> of a given task occurring in the context of that task.
>> Could be, but not necessarily the case. Eric has mentioned using elf
>> files for restart, and that's one way to go, but whether one central
>
> I'm not familiar with the details of this.

Well he wasn't specific and I'm not sure what his details were, I just
pictured it the way crack and other userspace c/r systems have worked,
where the checkpoint creates and ELF which you execute to restart the
task(set).

>> restart task sets up all the children or the children set themselves up
>> is yet another design point we haven't decided. I would think that

>> with a centralized restart it would be easier to assure for instance
>> that shared anon pages would be properly set up and shared, but since
>> you advocate each-task-starts-itself I trust zap must handle that.
>
> The main reason I think a task should setup itself, is because most of
> the setup requires that new resources be allocated, and the kernel is
> already centered around this approach that a task allocates for itself,
> not for another task. For instance, if you need to restore a VMA, you
> simply call mmap(), a new file, you call open() etc.

Agreed, it does seem cleaner, and if we go with the "sys_create_id()"
approach then clearly that's where we're aiming.

> Shared anon pages are one example of shared resources that may be used
> by multiple processes. Zap's approach is to have the "first" user (in
> the sense of the first time the resource is seen during checkpoint) do
> the actual restore, and place it in a global table, and then subsequent
> tasks will find it in the table and "map" it into their view.

Makes sense.

> Decentralizing also allow multiple tasks to restart concurrently.

Yes, but we lose that if we force create_with_pid() to be implemented
by setting /proc/sys/whatever/pid_min and max :)

> Are we ready to start concrete discussion on the architecture for the
> checkpoint/restart ? (and if so .. time to change the subject line).

Good news on this topic - unofficial word is that the containers
mini-summit at OLS has been approved. They don't yet know whether
it will be monday or tuesday, but hopefully this is enough information
early enough for anyone needing to make/change travel plans.

thanks,
-serge

>>> (I assume this is
>>> how restart will work). Therefore, it makes much sense that before every
>>> syscall that requires a pre-determined resource identifier (eg. clone,
>>> ipc, pty allocation), the task will place the desired value in "next_id"
>>> (and that will only be meaningful during restart) and invoke the said
>>> syscall. Voila.
>>>
>>> Note that the restart will "rebuild" the container's state (and the task
>>> state) as it reads in the data from some source. It is likely that not
>>> all data will be available when the first said syscall is about to be
>>> invoked, so you may not be able to feed everything ahead of time.

>>>
>>>
>>>> let the process pick up the objects ids as it needs them.
>>>> Of course, this would require to enhance the files formats, as well as
>>>> the way things are stored in the task_struct.
>>>>
>>>> Hope what I'm saying is not too stupid ;-) ?
>>>>
>>>> Regards,
>>>> Nadia
>>>>
>>>>>>> and why you'd
>>>>>>> want to write multiple identifiers in one go;
>>>>>>
>>>>>> I used multiple identifiers only for the pid values: this is because
>>>>>> when a new pid value is allocated for a process that belongs to nested
>>>>>> namespaces, the lower level upid nr values are allocated in a single
>>>>>> shot. (see alloc_pid()).
>>>>>>
>>>>>>> it seems to complicate the
>>>>>>> code and interface with minimal gain.
>>>>>>> In practice, a process will only do either one or the other, so a
>>>>>>> single
>>>>>>> file is enough (e.g. "next_id").
>>>>>>> Also, writing a single value at a time followed by the syscall is
>>>>>>> enough;
>>>>>>> it's definitely not a performance issue to have multiple calls.
>>>>>>> We assume the user/caller knows what she's doing, so no need to
>>>>>>> classify
>>>>>>> the identifier (that is, tell the kernel it's a pid, or an ipc id)
>>>>>>> ahead
>>>>>>> of time. The caller simply writes a value and then calls the relevant
>>>>>>> syscall, or otherwise the results may not be what she expected...
>>>>>>> If such context is expected to be required (although I don't see any
>>>>>>> at
>>>>>>> the moment), we can require that the user write "TYPE VALUE" pair to
>>>>>>> the "next_id" file.
>>>>>>
>>>>>> That's exactly what I wanted to avoid by creating 1 file per object.
>>>>>> Now, it's true that in a restart context where I guess that things
>>>>>> will be done synchronously, we could have a single next_id file.
>>>>>>
>>>>>>>> When one of these files (or both of them) is filled, a structure
>>>>>>>> pointed to
>>>>>>>> by the calling task struct is filled with these ids.
>>>>>>>>
>>>>>>>> Then, when the object is created, the id(s) present in that
>>>>>>>> structure are

>>>>>>>> used, instead of the default ones.
>>>>>>>>
>>>>>>>> The patches are against 2.6.25-rc3-mm1, in the following order:
>>>>>>>>
>>>>>>>> [PATCH 1/4] adds the procfs facility for next ipc to be created.
>>>>>>>> [PATCH 2/4] adds the procfs facility for next task to be forked.
>>>>>>>> [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.
>>>>>>>>
>>>>>>>> Cc-ing Pavel and Sukadev, since they are the pid namespace authors.
>>>>>>>>
>>>>>>>> Regards,
>>>>>>>> Nadia
>>>>>>>>
>>>>>>>> --
>>>>>>>>
>>>>>>>> --
>>>>>>>
>>>>>>>
>>>>>>>
>>>>>>
>>>>>> 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

---

Subject: Re: [RFC][PATCH 0/4] Object creation with a specified id
Posted by Nadia Derbey on Tue, 15 Apr 2008 10:30:26 GMT
View Forum Message <> Reply to Message

Nick Andrew wrote:

> 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.

Sorry for not answering earlier, I just saw your e-mail!

It's true that the way I've done things, the "create_with_id" doesn't
take into account multi-threaded apps, since "self" is related to the
thread group leader.

May be using something like /proc/self/task/<my_tid>/next_id would be
better, but I have to think more about it...

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 specified id
Posted by Oren Laadan on Tue, 15 Apr 2008 18:52:33 GMT
View Forum Message <> Reply to Message

Nadia Derbey wrote:
> Nick Andrew wrote:
>> 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.
>
>
> Sorry for not answering earlier, I just saw your e-mail!

[I too managed to miss that message].

>
> It's true that the way I've done things, the "create_with_id" doesn't
> take into account multi-threaded apps, since "self" is related to the
> thread group leader.
>
> May be using something like /proc/self/task/<my_tid>/next_id would be
> better, but I have to think more about it...

That /proc/self links to /proc/TGID slipped my mind. Definitely must
be done on a per-thread basis (and /proc/<TGID>/task/<PID>/next_id
will do the trick).

Oren.

>
> 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 specified id
Posted by Nadia Derbey on Fri, 18 Apr 2008 05:46:29 GMT
View Forum Message <> Reply to Message

Nick Andrew wrote:
> 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.

OK, race problem between threads is fixed. Thanks for finding the issue!

The new patch series is coming next.

Regards,
Nadia

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