Subject: [PATCH 0/10] Make bsd process accounting work in pid namespaces
Posted by Pavel Emelianov on Thu, 10 Apr 2008 08:17:47 GMT
View Forum Message <> Reply to Message

After I fixed access to task->tgid in kernel/acct.c, Oleg
pointed out some bad side effects with this accounting vs
pid namespaces interaction.

So here is the approach to make this accounting work with
pid namespaces properly.

The idea is simple - when task dies it accounts itself in
each namespace it is visible from. That was the summary,
the details are in patches.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

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

Subject: [PATCH 1/10] Bsdacct: rename acct_glbl structure
Posted by Pavel Emelianov on Thu, 10 Apr 2008 08:19:37 GMT
View Forum Message <> Reply to Message

It will be visible in pid_namespace.h file, so fixup its name first.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

---
 kernel/acct.c | 4 ++--
 1 files changed, 2 insertions(+), 2 deletions(-)

diff --git a/kernel/acct.c b/kernel/acct.c
index 91e1cfd..ee3e605 100644
--- a/kernel/acct.c
+++ b/kernel/acct.c
@@ -82,7 +82,7 @@ static void do_acct_process(struct pid_namespace *ns, struct file *);
  * can be placed in the same cache line as the lock. This primes
  * the cache line to have the data after getting the lock.
  */
-struct acct_glbs {
+struct bsd_acct_struct {
  spinlock_t lock;
  volatile int active;
  volatile int needcheck;
@@ -91,7 +91,7 @@ struct acct_glbs {

```
 struct timer_list timer;
};

-static struct acct_glbs acct_globals __cacheline_aligned =
+static struct bsd_acct_struct acct_globals __cacheline_aligned =
 {__SPIN_LOCK_UNLOCKED(acct_globals.lock)};

 /*
--
1.5.3.4
```

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

---

Subject: [PATCH 2/10] Pidns: use kzalloc to allocate new pid namespace
Posted by Pavel Emelianov on Thu, 10 Apr 2008 08:21:32 GMT
View Forum Message <> Reply to Message

It makes many fields initialization implicit, fixes void * = 0
(the ns->pidmap[i].page = 0) noise and will help in auto-setting
to NULL bsd-acct related field.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
---
 kernel/pid_namespace.c |    8 ++------
 1 files changed, 2 insertions(+), 6 deletions(-)

diff --git a/kernel/pid_namespace.c b/kernel/pid_namespace.c
index cb17497..06331cc 100644
--- a/kernel/pid_namespace.c
+++ b/kernel/pid_namespace.c
@@ -71,7 +71,7 @@ static struct pid_namespace *create_pid_namespace(unsigned int level)
  struct pid_namespace *ns;
  int i;

- ns = kmem_cache_alloc(pid_ns_cachep, GFP_KERNEL);
+ ns = kmem_cache_zalloc(pid_ns_cachep, GFP_KERNEL);
  if (ns == NULL)
   goto out;

@@ -84,17 +84,13 @@ static struct pid_namespace *create_pid_namespace(unsigned int level)
   goto out_free_map;

  kref_init(&ns->kref);
```

```
- ns->last_pid = 0;
- ns->child_reaper = NULL;
  ns->level = level;

  set_bit(0, ns->pidmap[0].page);
  atomic_set(&ns->pidmap[0].nr_free, BITS_PER_PAGE - 1);

- for (i = 1; i < PIDMAP_ENTRIES; i++) {
-  ns->pidmap[i].page = 0;
+ for (i = 1; i < PIDMAP_ENTRIES; i++)
   atomic_set(&ns->pidmap[i].nr_free, BITS_PER_PAGE);
- }

  return ns;

--
1.5.3.4
```

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

---

Subject: [PATCH 3/10] Pidns: add struct bsd_acct_struct *bacct field on pid_namespace

Posted by Pavel Emelianov on Thu, 10 Apr 2008 08:24:02 GMT

View Forum Message <> Reply to Message

It will be NULL automatically for all new namespaces.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
---
 include/linux/pid_namespace.h |    5 +++++
 1 files changed, 5 insertions(+), 0 deletions(-)

diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
index caff528..26e07d1 100644
--- a/include/linux/pid_namespace.h
+++ b/include/linux/pid_namespace.h
@@ -14,6 +14,8 @@ struct pidmap {

 #define PIDMAP_ENTRIES         ((PID_MAX_LIMIT + 8*PAGE_SIZE - 1)/PAGE_SIZE/8)

+struct bsd_acct_struct;
+
 struct pid_namespace {
```

```
  struct kref kref;
  struct pidmap pidmap[PIDMAP_ENTRIES];
@@ -25,6 +27,9 @@ struct pid_namespace {
 #ifdef CONFIG_PROC_FS
  struct vfsmount *proc_mnt;
 #endif
+#ifdef CONFIG_BSD_PROCESS_ACCT
+ struct bsd_acct_struct *bacct;
+#endif
 };

 extern struct pid_namespace init_pid_ns;
--
1.5.3.4
```

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

---

Subject: [PATCH 4/10] Bsdacct: fix bogus comment near acct_process
Posted by Pavel Emelianov on Thu, 10 Apr 2008 08:28:07 GMT
View Forum Message <> Reply to Message

It does not accept any arguments :)

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
---
 kernel/acct.c |    1 -
 1 files changed, 0 insertions(+), 1 deletions(-)

diff --git a/kernel/acct.c b/kernel/acct.c
index ee3e605..d9ee183 100644
--- a/kernel/acct.c
+++ b/kernel/acct.c
@@ -579,7 +579,6 @@ void acct_collect(long exitcode, int group_dead)

 /**
  * acct_process - now just a wrapper around do_acct_process
- * @exitcode: task exit code
  *
  * handles process accounting for an exiting task
  */
--
1.5.3.4
```

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

Subject: [PATCH 7/10] Bsdacct: stop using global bsd_acct_struct instance in internal functions
Posted by Pavel Emelianov on Thu, 10 Apr 2008 08:28:29 GMT
View Forum Message <> Reply to Message

This adds the appropriate pointer to all the internal (i.e. static)
functions that work with global acct instance. API calls pass this
global one to them by now.

Essentially this is a plain s/acct_globals./acct->/ over the file.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

---
 kernel/acct.c | 77 ++++++++++++++++++++++++++++++++----------------------
 1 files changed, 39 insertions(+), 38 deletions(-)

diff --git a/kernel/acct.c b/kernel/acct.c
index fc71c13..72d4760 100644
--- a/kernel/acct.c
+++ b/kernel/acct.c
@@ -75,7 +75,8 @@ int acct_parm[3] = {4, 2, 30};
 /*
  * External references and all of the globals.
  */
-static void do_acct_process(struct pid_namespace *ns, struct file *);
+static void do_acct_process(struct bsd_acct_struct *acct,
+ struct pid_namespace *ns, struct file *);

 /*
  * This structure is used so that all the data protected by lock
@@ -106,7 +107,7 @@ static void acct_timeout(unsigned long x)
 /*
  * Check the amount of free space and suspend/resume accordingly.
  */
-static int check_free_space(struct file *file)
+static int check_free_space(struct bsd_acct_struct *acct, struct file *file)
 {
  struct kstatfs sbuf;
  int res;
@@ -115,8 +116,8 @@ static int check_free_space(struct file *file)
  sector_t suspend;

```
  spin_lock(&acct_lock);
- res = acct_globals.active;
- if (!file || !acct_globals.needcheck)
+ res = acct->active;
+ if (!file || !acct->needcheck)
   goto out;
  spin_unlock(&acct_lock);

@@ -137,33 +138,33 @@ static int check_free_space(struct file *file)
   act = 0;

  /*
-  * If some joker switched acct_globals.file under us we'ld better be
+  * If some joker switched acct->file under us we'ld better be
   * silent and _not_ touch anything.
   */
  spin_lock(&acct_lock);
- if (file != acct_globals.file) {
+ if (file != acct->file) {
   if (act)
    res = act>0;
   goto out;
  }

- if (acct_globals.active) {
+ if (acct->active) {
   if (act < 0) {
-   acct_globals.active = 0;
+   acct->active = 0;
    printk(KERN_INFO "Process accounting paused\n");
   }
  } else {
   if (act > 0) {
-   acct_globals.active = 1;
+   acct->active = 1;
    printk(KERN_INFO "Process accounting resumed\n");
   }
  }

- del_timer(&acct_globals.timer);
- acct_globals.needcheck = 0;
- acct_globals.timer.expires = jiffies + ACCT_TIMEOUT*HZ;
- add_timer(&acct_globals.timer);
- res = acct_globals.active;
+ del_timer(&acct->timer);
+ acct->needcheck = 0;
+ acct->timer.expires = jiffies + ACCT_TIMEOUT*HZ;
```

```
+ add_timer(&acct->timer);
+ res = acct->active;
 out:
  spin_unlock(&acct_lock);
  return res;
@@ -175,34 +176,33 @@ out:
  *
  * NOTE: acct_lock MUST be held on entry and exit.
  */
-static void acct_file_reopen(struct file *file)
+static void acct_file_reopen(struct bsd_acct_struct *acct, struct file *file)
 {
  struct file *old_acct = NULL;
  struct pid_namespace *old_ns = NULL;

- if (acct_globals.file) {
-  old_acct = acct_globals.file;
-  old_ns = acct_globals.ns;
-  del_timer(&acct_globals.timer);
-  acct_globals.active = 0;
-  acct_globals.needcheck = 0;
-  acct_globals.file = NULL;
+ if (acct->file) {
+  old_acct = acct->file;
+  old_ns = acct->ns;
+  del_timer(&acct->timer);
+  acct->active = 0;
+  acct->needcheck = 0;
+  acct->file = NULL;
  }
  if (file) {
-  acct_globals.file = file;
-  acct_globals.ns = get_pid_ns(task_active_pid_ns(current));
-  acct_globals.needcheck = 0;
-  acct_globals.active = 1;
+  acct->file = file;
+  acct->ns = get_pid_ns(task_active_pid_ns(current));
+  acct->needcheck = 0;
+  acct->active = 1;
   /* It's been deleted if it was used before so this is safe */
-  setup_timer(&acct_globals.timer, acct_timeout,
-    (unsigned long)&acct_globals);
-  acct_globals.timer.expires = jiffies + ACCT_TIMEOUT*HZ;
-  add_timer(&acct_globals.timer);
+  setup_timer(&acct->timer, acct_timeout, (unsigned long)acct);
+  acct->timer.expires = jiffies + ACCT_TIMEOUT*HZ;
+  add_timer(&acct->timer);
  }
```

```
  if (old_acct) {
   mnt_unpin(old_acct->f_path.mnt);
   spin_unlock(&acct_lock);
-  do_acct_process(old_ns, old_acct);
+  do_acct_process(acct, old_ns, old_acct);
   filp_close(old_acct, NULL);
   put_pid_ns(old_ns);
   spin_lock(&acct_lock);
@@ -237,7 +237,7 @@ static int acct_on(char *name)

  spin_lock(&acct_lock);
  mnt_pin(file->f_path.mnt);
- acct_file_reopen(file);
+ acct_file_reopen(&acct_globals, file);
  spin_unlock(&acct_lock);

  mntput(file->f_path.mnt); /* it's pinned, now give up active reference */
@@ -273,7 +273,7 @@ asmlinkage long sys_acct(const char __user *name)
   error = security_acct(NULL);
   if (!error) {
    spin_lock(&acct_lock);
-   acct_file_reopen(NULL);
+   acct_file_reopen(&acct_globals, NULL);
    spin_unlock(&acct_lock);
   }
  }
@@ -291,7 +291,7 @@ void acct_auto_close_mnt(struct vfsmount *m)
 {
  spin_lock(&acct_lock);
  if (acct_globals.file && acct_globals.file->f_path.mnt == m)
-  acct_file_reopen(NULL);
+  acct_file_reopen(&acct_globals, NULL);
  spin_unlock(&acct_lock);
 }

@@ -307,7 +307,7 @@ void acct_auto_close(struct super_block *sb)
  spin_lock(&acct_lock);
  if (acct_globals.file &&
      acct_globals.file->f_path.mnt->mnt_sb == sb) {
-  acct_file_reopen(NULL);
+  acct_file_reopen(&acct_globals, NULL);
  }
  spin_unlock(&acct_lock);
 }
@@ -426,7 +426,8 @@ static u32 encode_float(u64 value)
 /*
  *  do_acct_process does all actual work. Caller holds the reference to file.
  */
```

```
-static void do_acct_process(struct pid_namespace *ns, struct file *file)
+static void do_acct_process(struct bsd_acct_struct *acct,
+  struct pid_namespace *ns, struct file *file)
 {
  struct pacct_struct *pacct = &current->signal->pacct;
  acct_t ac;
@@ -441,7 +442,7 @@ static void do_acct_process(struct pid_namespace *ns, struct file *file)
   * First check to see if there is enough free_space to continue
   * the process accounting system.
   */
- if (!check_free_space(file))
+ if (!check_free_space(acct, file))
   return;

  /*
@@ -604,7 +605,7 @@ void acct_process(void)
  ns = get_pid_ns(acct_globals.ns);
  spin_unlock(&acct_lock);

- do_acct_process(ns, file);
+ do_acct_process(&acct_globals, ns, file);
  fput(file);
  put_pid_ns(ns);
 }
--
1.5.3.4
```

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

---

Subject: [PATCH 6/10] Bsdacct: make the acct_lock global
Posted by Pavel Emelianov on Thu, 10 Apr 2008 08:28:58 GMT
View Forum Message <> Reply to Message

Don't use per-bsd-acct-struct lock, but work with a global
one. This lock is taken for short periods, so it's not going
to be a bottleneck, but it will allow us to easily avoid
many locking difficulties in the future.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
---
 kernel/acct.c |   42 ++++++++++++++++++++++++++----------------
 1 files changed, 21 insertions(+), 21 deletions(-)
```

```
diff --git a/kernel/acct.c b/kernel/acct.c
index 05f8bc0..fc71c13 100644
--- a/kernel/acct.c
+++ b/kernel/acct.c
@@ -83,7 +83,6 @@ static void do_acct_process(struct pid_namespace *ns, struct file *);
  * the cache line to have the data after getting the lock.
  */
 struct bsd_acct_struct {
- spinlock_t  lock;
  volatile int  active;
  volatile int  needcheck;
  struct file  *file;
@@ -91,8 +90,9 @@ struct bsd_acct_struct {
  struct timer_list timer;
 };

-static struct bsd_acct_struct acct_globals __cacheline_aligned =
- {__SPIN_LOCK_UNLOCKED(acct_globals.lock)};
+static DEFINE_SPINLOCK(acct_lock);
+
+static struct bsd_acct_struct acct_globals __cacheline_aligned;

 /*
  * Called whenever the timer says to check the free space.
@@ -114,11 +114,11 @@ static int check_free_space(struct file *file)
  sector_t resume;
  sector_t suspend;

- spin_lock(&acct_globals.lock);
+ spin_lock(&acct_lock);
  res = acct_globals.active;
  if (!file || !acct_globals.needcheck)
   goto out;
- spin_unlock(&acct_globals.lock);
+ spin_unlock(&acct_lock);

  /* May block */
  if (vfs_statfs(file->f_path.dentry, &sbuf))
@@ -140,7 +140,7 @@ static int check_free_space(struct file *file)
   * If some joker switched acct_globals.file under us we'ld better be
   * silent and _not_ touch anything.
   */
- spin_lock(&acct_globals.lock);
+ spin_lock(&acct_lock);
  if (file != acct_globals.file) {
   if (act)
    res = act>0;
@@ -165,7 +165,7 @@ static int check_free_space(struct file *file)
```

```
  add_timer(&acct_globals.timer);
  res = acct_globals.active;
 out:
- spin_unlock(&acct_globals.lock);
+ spin_unlock(&acct_lock);
  return res;
 }

@@ -173,7 +173,7 @@ out:
  * Close the old accounting file (if currently open) and then replace
  * it with file (if non-NULL).
  *
- * NOTE: acct_globals.lock MUST be held on entry and exit.
+ * NOTE: acct_lock MUST be held on entry and exit.
  */
 static void acct_file_reopen(struct file *file)
 {
@@ -201,11 +201,11 @@ static void acct_file_reopen(struct file *file)
  }
  if (old_acct) {
   mnt_unpin(old_acct->f_path.mnt);
-  spin_unlock(&acct_globals.lock);
+  spin_unlock(&acct_lock);
   do_acct_process(old_ns, old_acct);
   filp_close(old_acct, NULL);
   put_pid_ns(old_ns);
-  spin_lock(&acct_globals.lock);
+  spin_lock(&acct_lock);
  }
 }

@@ -235,10 +235,10 @@ static int acct_on(char *name)
   return error;
  }

- spin_lock(&acct_globals.lock);
+ spin_lock(&acct_lock);
  mnt_pin(file->f_path.mnt);
  acct_file_reopen(file);
- spin_unlock(&acct_globals.lock);
+ spin_unlock(&acct_lock);

  mntput(file->f_path.mnt); /* it's pinned, now give up active reference */

@@ -272,9 +272,9 @@ asmlinkage long sys_acct(const char __user *name)
  } else {
   error = security_acct(NULL);
   if (!error) {
```

```
-   spin_lock(&acct_globals.lock);
+   spin_lock(&acct_lock);
    acct_file_reopen(NULL);
-   spin_unlock(&acct_globals.lock);
+   spin_unlock(&acct_lock);
   }
  }
  return error;
@@ -289,10 +289,10 @@ asmlinkage long sys_acct(const char __user *name)
  */
 void acct_auto_close_mnt(struct vfsmount *m)
 {
- spin_lock(&acct_globals.lock);
+ spin_lock(&acct_lock);
  if (acct_globals.file && acct_globals.file->f_path.mnt == m)
   acct_file_reopen(NULL);
- spin_unlock(&acct_globals.lock);
+ spin_unlock(&acct_lock);
 }

 /**
@@ -304,12 +304,12 @@ void acct_auto_close_mnt(struct vfsmount *m)
  */
 void acct_auto_close(struct super_block *sb)
 {
- spin_lock(&acct_globals.lock);
+ spin_lock(&acct_lock);
  if (acct_globals.file &&
      acct_globals.file->f_path.mnt->mnt_sb == sb) {
   acct_file_reopen(NULL);
  }
- spin_unlock(&acct_globals.lock);
+ spin_unlock(&acct_lock);
 }

 /*
@@ -594,15 +594,15 @@ void acct_process(void)
  if (!acct_globals.file)
   return;

- spin_lock(&acct_globals.lock);
+ spin_lock(&acct_lock);
  file = acct_globals.file;
  if (unlikely(!file)) {
-  spin_unlock(&acct_globals.lock);
+  spin_unlock(&acct_lock);
   return;
  }
```

```
  get_file(file);
  ns = get_pid_ns(acct_globals.ns);
- spin_unlock(&acct_globals.lock);
+ spin_unlock(&acct_lock);

  do_acct_process(ns, file);
  fput(file);
--
1.5.3.4
```

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

---

Subject: [PATCH 5/10] Bsdacct: make check timer accept an argument
Posted by xemul on Thu, 10 Apr 2008 08:29:01 GMT
View Forum Message <> Reply to Message

We're going to have many bsd_acct_struct instances, not just
one, so the timer (currently working with a global one) has
to know which one to work with.

Use a handy setup_timer macro for it (thanks to Oleg for one).

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
---
 kernel/acct.c |    9 +++++----
 1 files changed, 5 insertions(+), 4 deletions(-)

diff --git a/kernel/acct.c b/kernel/acct.c
index d9ee183..05f8bc0 100644
--- a/kernel/acct.c
+++ b/kernel/acct.c
@@ -97,9 +97,10 @@ static struct bsd_acct_struct acct_globals __cacheline_aligned =
 /*
  * Called whenever the timer says to check the free space.
  */
-static void acct_timeout(unsigned long unused)
+static void acct_timeout(unsigned long x)
 {
- acct_globals.needcheck = 1;
+ struct bsd_acct_struct *acct = (struct bsd_acct_struct *)x;
+ acct->needcheck = 1;
 }
```

```
 /*
@@ -193,8 +194,8 @@ static void acct_file_reopen(struct file *file)
   acct_globals.needcheck = 0;
   acct_globals.active = 1;
   /* It's been deleted if it was used before so this is safe */
-  init_timer(&acct_globals.timer);
-  acct_globals.timer.function = acct_timeout;
+  setup_timer(&acct_globals.timer, acct_timeout,
+    (unsigned long)&acct_globals);
   acct_globals.timer.expires = jiffies + ACCT_TIMEOUT*HZ;
   add_timer(&acct_globals.timer);
  }
--
1.5.3.4
```

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

Subject: [PATCH 8/10] Bsdacct: make bsd_acct_struct per pid namespace
Posted by Pavel Emelianov on Thu, 10 Apr 2008 08:31:45 GMT

View Forum Message <> Reply to Message

Allocate the structure on the first call to sys_acct(). After
this each namespace, that ordered the accounting, will live
with this structure till its own death.

Two notes
- routines, that close the accounting on fs umount time use
  the init_pid_ns's acct by now;
- accounting routine accounts to dying task's namespace
  (also by now).

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
---
 include/linux/acct.h   |    3 ++
 kernel/acct.c          |   84 +++++++++++++++++++++++++++++++++++++++++++++----------
 kernel/pid_namespace.c |    2 +
 3 files changed, 71 insertions(+), 18 deletions(-)

diff --git a/include/linux/acct.h b/include/linux/acct.h
index e8cae54..882dc72 100644
--- a/include/linux/acct.h
+++ b/include/linux/acct.h
@@ -120,17 +120,20 @@ struct acct_v3
```

```
 struct vfsmount;
 struct super_block;
 struct pacct_struct;
+struct pid_namespace;
 extern void acct_auto_close_mnt(struct vfsmount *m);
 extern void acct_auto_close(struct super_block *sb);
 extern void acct_init_pacct(struct pacct_struct *pacct);
 extern void acct_collect(long exitcode, int group_dead);
 extern void acct_process(void);
+extern void acct_exit_ns(struct pid_namespace *);
 #else
 #define acct_auto_close_mnt(x) do { } while (0)
 #define acct_auto_close(x) do { } while (0)
 #define acct_init_pacct(x) do { } while (0)
 #define acct_collect(x,y) do { } while (0)
 #define acct_process()  do { } while (0)
+#define acct_exit_ns(ns) do { } while (0)
 #endif

 /*
diff --git a/kernel/acct.c b/kernel/acct.c
index 72d4760..febbbc6 100644
--- a/kernel/acct.c
+++ b/kernel/acct.c
@@ -93,8 +93,6 @@ struct bsd_acct_struct {

 static DEFINE_SPINLOCK(acct_lock);

-static struct bsd_acct_struct acct_globals __cacheline_aligned;
-
 /*
  * Called whenever the timer says to check the free space.
  */
@@ -176,7 +174,8 @@ out:
  *
  * NOTE: acct_lock MUST be held on entry and exit.
  */
-static void acct_file_reopen(struct bsd_acct_struct *acct, struct file *file)
+static void acct_file_reopen(struct bsd_acct_struct *acct, struct file *file,
+  struct pid_namespace *ns)
 {
  struct file *old_acct = NULL;
  struct pid_namespace *old_ns = NULL;
@@ -188,10 +187,11 @@ static void acct_file_reopen(struct bsd_acct_struct *acct, struct file
*file)
   acct->active = 0;
   acct->needcheck = 0;
   acct->file = NULL;
```

```
+  acct->ns = NULL;
  }
  if (file) {
   acct->file = file;
-  acct->ns = get_pid_ns(task_active_pid_ns(current));
+  acct->ns = ns;
   acct->needcheck = 0;
   acct->active = 1;
   /* It's been deleted if it was used before so this is safe */
@@ -204,7 +204,6 @@ static void acct_file_reopen(struct bsd_acct_struct *acct, struct file *file)
   spin_unlock(&acct_lock);
   do_acct_process(acct, old_ns, old_acct);
   filp_close(old_acct, NULL);
-  put_pid_ns(old_ns);
   spin_lock(&acct_lock);
  }
 }
@@ -213,6 +212,8 @@ static int acct_on(char *name)
 {
  struct file *file;
  int error;
+ struct pid_namespace *ns;
+ struct bsd_acct_struct *acct = NULL;

  /* Difference from BSD - they don't do O_APPEND */
  file = filp_open(name, O_WRONLY|O_APPEND|O_LARGEFILE, 0);
@@ -229,18 +230,34 @@ static int acct_on(char *name)
   return -EIO;
  }

+ ns = task_active_pid_ns(current);
+ if (ns->bacct == NULL) {
+  acct = kzalloc(sizeof(struct bsd_acct_struct), GFP_KERNEL);
+  if (acct == NULL) {
+   filp_close(file, NULL);
+   return -ENOMEM;
+  }
+ }
+
  error = security_acct(file);
  if (error) {
+  kfree(acct);
   filp_close(file, NULL);
   return error;
  }

  spin_lock(&acct_lock);
+ if (ns->bacct == NULL) {
```

```
+  ns->bacct = acct;
+  acct = NULL;
+ }
+
  mnt_pin(file->f_path.mnt);
- acct_file_reopen(&acct_globals, file);
+ acct_file_reopen(ns->bacct, file, ns);
  spin_unlock(&acct_lock);

  mntput(file->f_path.mnt); /* it's pinned, now give up active reference */
+ kfree(acct);

  return 0;
 }
@@ -270,10 +287,16 @@ asmlinkage long sys_acct(const char __user *name)
   error = acct_on(tmp);
   putname(tmp);
  } else {
+  struct bsd_acct_struct *acct;
+
+  acct = task_active_pid_ns(current)->bacct;
+  if (acct == NULL)
+   return 0;
+
   error = security_acct(NULL);
   if (!error) {
    spin_lock(&acct_lock);
-   acct_file_reopen(&acct_globals, NULL);
+   acct_file_reopen(acct, NULL, NULL);
    spin_unlock(&acct_lock);
   }
  }
@@ -289,9 +312,15 @@ asmlinkage long sys_acct(const char __user *name)
  */
 void acct_auto_close_mnt(struct vfsmount *m)
 {
+ struct bsd_acct_struct *acct;
+
+ acct = init_pid_ns.bacct;
+ if (acct == NULL)
+  return;
+
  spin_lock(&acct_lock);
- if (acct_globals.file && acct_globals.file->f_path.mnt == m)
-  acct_file_reopen(&acct_globals, NULL);
+ if (acct->file && acct->file->f_path.mnt == m)
+  acct_file_reopen(acct, NULL, NULL);
  spin_unlock(&acct_lock);
```

```
 }

@@ -304,10 +333,29 @@ void acct_auto_close_mnt(struct vfsmount *m)
  */
 void acct_auto_close(struct super_block *sb)
 {
+ struct bsd_acct_struct *acct;
+
+ acct = init_pid_ns.bacct;
+ if (acct == NULL)
+  return;
+
  spin_lock(&acct_lock);
- if (acct_globals.file &&
-     acct_globals.file->f_path.mnt->mnt_sb == sb) {
-  acct_file_reopen(&acct_globals, NULL);
+ if (acct->file && acct->file->f_path.mnt->mnt_sb == sb)
+  acct_file_reopen(acct, NULL, NULL);
+ spin_unlock(&acct_lock);
+}
+
+void acct_exit_ns(struct pid_namespace *ns)
+{
+ struct bsd_acct_struct *acct;
+
+ spin_lock(&acct_lock);
+ acct = ns->bacct;
+ if (acct != NULL) {
+  if (acct->file != NULL)
+   acct_file_reopen(acct, NULL, NULL);
+
+  kfree(acct);
  }
  spin_unlock(&acct_lock);
 }
@@ -587,25 +635,25 @@ void acct_collect(long exitcode, int group_dead)
 void acct_process(void)
 {
  struct file *file = NULL;
- struct pid_namespace *ns;
+ struct pid_namespace *ns = task_active_pid_ns(current);
+ struct bsd_acct_struct *acct;

+ acct = ns->bacct;
  /*
   * accelerate the common fastpath:
   */
- if (!acct_globals.file)
```

```
+ if (!acct || !acct->file)
   return;

  spin_lock(&acct_lock);
- file = acct_globals.file;
+ file = acct->file;
  if (unlikely(!file)) {
   spin_unlock(&acct_lock);
   return;
  }
  get_file(file);
- ns = get_pid_ns(acct_globals.ns);
  spin_unlock(&acct_lock);

- do_acct_process(&acct_globals, ns, file);
+ do_acct_process(acct, ns, file);
  fput(file);
- put_pid_ns(ns);
 }
diff --git a/kernel/pid_namespace.c b/kernel/pid_namespace.c
index 06331cc..ea567b7 100644
--- a/kernel/pid_namespace.c
+++ b/kernel/pid_namespace.c
@@ -12,6 +12,7 @@
 #include <linux/pid_namespace.h>
 #include <linux/syscalls.h>
 #include <linux/err.h>
+#include <linux/acct.h>

 #define BITS_PER_PAGE  (PAGE_SIZE*8)

@@ -181,6 +182,7 @@ void zap_pid_ns_processes(struct pid_namespace *pid_ns)

  /* Child reaper for the pid namespace is going away */
  pid_ns->child_reaper = NULL;
+ acct_exit_ns(pid_ns);
  return;
 }

--
1.5.3.4
```

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

Subject: [PATCH 9/10] Bsdacct: stop accounting for all namespaces on umount time
Posted by Pavel Emelianov on Thu, 10 Apr 2008 08:37:14 GMT
View Forum Message <> Reply to Message

All the bsd_acct_strcts with opened accounting are linked into
a global list. So, the acct_auto_close(_mnt) walks one and drops
the accounting for each.

BTW, it worth dropping the acct->file != NULL check from these
routines, but I noticed this only when started sending the set :(
... Will do it in the v2.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

---
 kernel/acct.c | 28 ++++++++++++++++++----------
 1 files changed, 16 insertions(+), 12 deletions(-)

diff --git a/kernel/acct.c b/kernel/acct.c
index febbbc6..7fc9f9d 100644
--- a/kernel/acct.c
+++ b/kernel/acct.c
@@ -89,9 +89,11 @@ struct bsd_acct_struct {
  struct file *file;
  struct pid_namespace *ns;
  struct timer_list timer;
+ struct list_head list;
 };

 static DEFINE_SPINLOCK(acct_lock);
+static LIST_HEAD(acct_list);

 /*
  * Called whenever the timer says to check the free space.
@@ -188,12 +190,14 @@ static void acct_file_reopen(struct bsd_acct_struct *acct, struct file *file,
   acct->needcheck = 0;
   acct->file = NULL;
   acct->ns = NULL;
+ list_del(&acct->list);
  }
  if (file) {
   acct->file = file;
   acct->ns = ns;
   acct->needcheck = 0;
   acct->active = 1;
+ list_add(&acct->list, &acct_list);
   /* It's been deleted if it was used before so this is safe */

```
   setup_timer(&acct->timer, acct_timeout, (unsigned long)acct);
   acct->timer.expires = jiffies + ACCT_TIMEOUT*HZ;
@@ -314,13 +318,13 @@ void acct_auto_close_mnt(struct vfsmount *m)
 {
  struct bsd_acct_struct *acct;

- acct = init_pid_ns.bacct;
- if (acct == NULL)
-  return;
-
  spin_lock(&acct_lock);
- if (acct->file && acct->file->f_path.mnt == m)
-  acct_file_reopen(acct, NULL, NULL);
+restart:
+ list_for_each_entry(acct, &acct_list, list)
+  if (acct->file && acct->file->f_path.mnt == m) {
+   acct_file_reopen(acct, NULL, NULL);
+   goto restart;
+  }
  spin_unlock(&acct_lock);
 }

@@ -335,13 +339,13 @@ void acct_auto_close(struct super_block *sb)
 {
  struct bsd_acct_struct *acct;

- acct = init_pid_ns.bacct;
- if (acct == NULL)
-  return;
-
  spin_lock(&acct_lock);
- if (acct->file && acct->file->f_path.mnt->mnt_sb == sb)
-  acct_file_reopen(acct, NULL, NULL);
+restart:
+ list_for_each_entry(acct, &acct_list, list)
+  if (acct->file && acct->file->f_path.mnt->mnt_sb == sb) {
+   acct_file_reopen(acct, NULL, NULL);
+   goto restart;
+  }
  spin_unlock(&acct_lock);
 }

--
1.5.3.4
```

Containers mailing list
Containers@lists.linux-foundation.org

Subject: [PATCH 10/10] Bsdacct: account task in each namespace is is visible from
Posted by Pavel Emelianov on Thu, 10 Apr 2008 08:38:19 GMT
View Forum Message <> Reply to Message

This just makes the acct_proces walk the pid namespaces
from current up to the top and account a task in each
with the accounting turned on.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

---
 kernel/acct.c | 21 +++++++++++++++-------
 1 files changed, 14 insertions(+), 7 deletions(-)

diff --git a/kernel/acct.c b/kernel/acct.c
index 7fc9f9d..0feba97 100644
--- a/kernel/acct.c
+++ b/kernel/acct.c
@@ -631,15 +631,9 @@ void acct_collect(long exitcode, int group_dead)
  spin_unlock_irq(&current->sighand->siglock);
 }

-/**
- * acct_process - now just a wrapper around do_acct_process
- *
- * handles process accounting for an exiting task
- */
-void acct_process(void)
+static void acct_process_in_ns(struct pid_namespace *ns)
 {
  struct file *file = NULL;
- struct pid_namespace *ns = task_active_pid_ns(current);
  struct bsd_acct_struct *acct;

  acct = ns->bacct;
@@ -661,3 +655,16 @@ void acct_process(void)
  do_acct_process(acct, ns, file);
  fput(file);
 }
+
+/**
+ * acct_process - now just a wrapper around do_acct_process
+ *
+ * handles process accounting for an exiting task
+ */

```
+void acct_process(void)
+{
+ struct pid_namespace *ns;
+
+ for (ns = task_active_pid_ns(current); ns != NULL; ns = ns->parent)
+  acct_process_in_ns(ns);
+}
--
1.5.3.4
```

_____

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