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
```