Subject: [PATCH 3/4] proc: simplify remove_proc_entry() wrt locking
Posted by Alexey Dobriyan on Fri, 16 Nov 2007 15:10:15 GMT
View Forum Message <> Reply to Message

We can take proc_subdir_lock for duration of list searching and removing
from lists only. It can't hurt -- we can gather any amount of looked up
PDEs right after proc_subdir_lock droppage in proc_lookup() anyway.
Current code should already deal with this correctly.

Also this should make code more undestandable:
* original looks like a loop, however, it's a loop with unconditional
  trailing "break;" -- not loop at all.
* more explicit statement that proc_subdir_lock protects only ->subdir lists.

Signed-off-by: Alexey Dobriyan <adobriyan@sw.ru>

```
 fs/proc/generic.c |   67 ++++++++++++++++++++++++++++----------------------------
 1 file changed, 32 insertions(+), 35 deletions(-)

--- a/fs/proc/generic.c
+++ b/fs/proc/generic.c
@@ -686,12 +686,12 @@ void free_proc_entry(struct proc_dir_entry *de)
 void remove_proc_entry(const char *name, struct proc_dir_entry *parent)
 {
  struct proc_dir_entry **p;
- struct proc_dir_entry *de;
+ struct proc_dir_entry *de = NULL;
  const char *fn = name;
  int len;

  if (!parent && xlate_proc_name(name, &parent, &fn) != 0)
-  goto out;
+  return;
  len = strlen(fn);

  spin_lock(&proc_subdir_lock);
@@ -701,45 +701,42 @@ void remove_proc_entry(const char *name, struct proc_dir_entry
*parent)
   de = *p;
   *p = de->next;
   de->next = NULL;
+ }
+ spin_unlock(&proc_subdir_lock);
+ if (!de)
+  return;

-  spin_lock(&de->pde_unload_lock);
```

```
-  /*
-   * Stop accepting new callers into module. If you're
-   * dynamically allocating ->proc_fops, save a pointer somewhere.
-   */
-  de->proc_fops = NULL;
-  /* Wait until all existing callers into module are done. */
-  if (de->pde_users > 0) {
-   DECLARE_COMPLETION_ONSTACK(c);
+ spin_lock(&de->pde_unload_lock);
+ /*
+  * Stop accepting new callers into module. If you're
+  * dynamically allocating ->proc_fops, save a pointer somewhere.
+  */
+ de->proc_fops = NULL;
+ /* Wait until all existing callers into module are done. */
+ if (de->pde_users > 0) {
+  DECLARE_COMPLETION_ONSTACK(c);

-   if (!de->pde_unload_completion)
-    de->pde_unload_completion = &c;
+  if (!de->pde_unload_completion)
+   de->pde_unload_completion = &c;

-   spin_unlock(&de->pde_unload_lock);
-   spin_unlock(&proc_subdir_lock);
+  spin_unlock(&de->pde_unload_lock);

-   wait_for_completion(de->pde_unload_completion);
+  wait_for_completion(de->pde_unload_completion);

-   spin_lock(&proc_subdir_lock);
-   goto continue_removing;
-  }
-  spin_unlock(&de->pde_unload_lock);
+  goto continue_removing;
+ }
+ spin_unlock(&de->pde_unload_lock);

 continue_removing:
-  if (S_ISDIR(de->mode))
-   parent->nlink--;
-  de->nlink = 0;
-  WARN_ON(de->subdir);
-  if (!atomic_read(&de->count))
-   free_proc_entry(de);
-  else {
-   de->deleted = 1;
-   printk("remove_proc_entry: %s/%s busy, count=%d\n",
```

```
-    parent->name, de->name, atomic_read(&de->count));
-  }
-  break;
+ if (S_ISDIR(de->mode))
+  parent->nlink--;
+ de->nlink = 0;
+ WARN_ON(de->subdir);
+ if (!atomic_read(&de->count))
+  free_proc_entry(de);
+ else {
+  de->deleted = 1;
+  printk("remove_proc_entry: %s/%s busy, count=%d\n",
+   parent->name, de->name, atomic_read(&de->count));
  }
- spin_unlock(&proc_subdir_lock);
-out:
- return;
 }
```