Subject: Re: [PATCH v5] Fix rmmod/read/write races in /proc entries
Posted by Andrew Morton on Fri, 16 Mar 2007 01:53:04 GMT
View Forum Message <> Reply to Message

On Sun, 11 Mar 2007 20:04:56 +0300 Alexey Dobriyan <adobriyan@sw.ru> wrote:

```
> Differences from version 4:
>  Updated in-code comments. Largely rewritten changelog.
>  Lockdep please. --akpm
>  ->read_proc, ->write_proc aren't special, Extend protection to
>  most methods for regular /proc files. Mentioned by viro.
> Differences from version 3:
>  Use completion instead of unlock/schedule/lock
>  Move refcount waiting business after removing PDE from lists,
>  so that *cough* possible concurrent remove_proc_entry() will
>  work.
```

My, what a lot of code you have here. I note that nobody can be assed even reviewing it. Now why is that?

```
> Fix following races:
> ===========================================
> 1. Write via ->write_proc sleeps in copy_from_user(). Module disappears
>    meanwhile. Or, more generically, system call done on /proc file, method
>    supplied by module is called, module dissapeares meanwhile.
>
>    pde = create_proc_entry()
>    if (!pde)
>  return -ENOMEM;
>    pde->write_proc = ...
>     open
>     write
>     copy_from_user
>    pde = create_proc_entry();
>    if (!pde) {
>  remove_proc_entry();
>  return -ENOMEM;
>  /* module unloaded */
>    }
```

We usually fix that race by pinning the module: make whoever registered the proc entries also register their THIS_MODULE, do a try_module_get() on it before we start to play with data structures which the module owns.

Can we do that here?

And is the above race fix related to the below one in any fashion?

```
> ==========================================
> 2. bogo-revoke aka proc_kill_inodes()
>
>   remove_proc_entry  vfs_read
>   proc_kill_inodes  [check ->f_op validness]
>     [check ->f_op->read validness]
>     [verify_area, security permissions checks]
>  ->f_op = NULL;
>     if (file->f_op->read)
>      /* ->f_op dereference, boom */
```

So you fixed this via sort-of-refcounting on pde->pde_users.

hmm.

```
> NOTE, NOTE, NOTE: file_operations are proxied for regular files only. Let's
> see how this scheme behaves, then extend if needed for directories.
> Directories creators in /proc only set ->owner for them, so proxying for
> directories may be unneeded.
>
> NOTE, NOTE, NOTE: methods being proxied are ->llseek, ->read, ->write,
> ->poll, ->unlocked_ioctl, ->ioctl, ->compat_ioctl, ->open, ->release.
> If your in-tree module uses something else, yell on me. Full audit pending.
>
> Signed-off-by: Alexey Dobriyan <adobriyan@sw.ru>
> ---
>
>  fs/proc/generic.c       |   32 +++++
>  fs/proc/inode.c         |  279 +++++++++++++++++++++++++++++++++++++++++++++++-
>  include/linux/proc_fs.h |   13 ++
>  3 files changed, 321 insertions(+), 3 deletions(-)
>
> --- a/fs/proc/generic.c
> +++ b/fs/proc/generic.c
> @@ -20,6 +20,7 @@ #include <linux/idr.h>
>  #include <linux/namei.h>
>  #include <linux/bitops.h>
>  #include <linux/spinlock.h>
> +#include <linux/completion.h>
>  #include <asm/uaccess.h>
>
>  #include "internal.h"
> @@ -613,6 +614,9 @@ static struct proc_dir_entry *proc_creat
>   ent->namelen = len;
>   ent->mode = mode;
>   ent->nlink = nlink;
> + ent->pde_users = 0;
> + spin_lock_init(&ent->pde_unload_lock);
> + ent->pde_unload_completion = NULL;
>   out:
>   return ent;
>  }
> @@ -734,9 +738,35 @@ void remove_proc_entry(const char *name,
>    de = *p;
>    *p = de->next;
>    de->next = NULL;
> +
> +  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;
> +
> +   spin_unlock(&de->pde_unload_lock);
> +   spin_unlock(&proc_subdir_lock);
> +
> +   wait_for_completion(de->pde_unload_completion);
> +
> +   spin_lock(&proc_subdir_lock);
> +   goto continue_removing;
> +  }
> +  spin_unlock(&de->pde_unload_lock);
> +
> +continue_removing:
>    if (S_ISDIR(de->mode))
>     parent->nlink--;
> -  proc_kill_inodes(de);
> +  if (!S_ISREG(de->mode))
> +   proc_kill_inodes(de);
>    de->nlink = 0;
>    WARN_ON(de->subdir);
>    if (!atomic_read(&de->count))
> --- a/fs/proc/inode.c
> +++ b/fs/proc/inode.c
> @@ -142,6 +142,277 @@ static const struct super_operations pro
>   .remount_fs = proc_remount,
>  };
>
> +static loff_t proc_reg_llseek(struct file *file, loff_t offset, int whence)
> +{
> + struct proc_dir_entry *pde = PDE(file->f_path.dentry->d_inode);
> + loff_t rv = -EINVAL;
> + loff_t (*llseek)(struct file *, loff_t, int);
> +
> + spin_lock(&pde->pde_unload_lock);
> + /*
> +  * remove_proc_entry() is going to delete PDE (as part of module
> +  * cleanup sequence). No new callers into module allowed.
> +  */
> + if (!pde->proc_fops)
> +  goto out_unlock;
> + /*
> +  * Bump refcount so that remove_proc_entry will wail for ->llseek to
> +  * complete.
> +  */
> + pde->pde_users++;
> + /*
> +  * Save function pointer under lock, to protect against ->proc_fops
> +  * NULL'ifying right after ->pde_unload_lock is dropped.
> +  */
> + llseek = pde->proc_fops->llseek;
> + spin_unlock(&pde->pde_unload_lock);
> +
> + if (llseek)
> +  rv = llseek(file, offset, whence);
> +
> + spin_lock(&pde->pde_unload_lock);
> + pde->pde_users--;
> + if (pde->pde_unload_completion && pde->pde_users == 0)
> +  complete(pde->pde_unload_completion);
> +out_unlock:
> + spin_unlock(&pde->pde_unload_lock);
```

The above six lines happen rather a lot - perhaps it could be placed in a helper funtion?