
Subject: [PATCH] Fix rmmod/read/write races in /proc entries
Posted by [adobriyan](#) on Mon, 15 Jan 2007 15:33:41 GMT
[View Forum Message](#) <> [Reply to Message](#)

```
-----  
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 */  
}  
    *boom*
```

```
-----  
pde = create_proc_entry();  
if (pde) {  
    /* which dereferences ->data */  
    pde->write_proc = ...;  
        open  
        write  
    pde->data = ...;  
}  
-----
```

The following plan is going to be executed (as per Al Viro's explanations):

PDE gets atomic counter counting reads and writes in progress via ->read_proc, ->write_proc, ->get_info . Generic proc code will bump PDE's counter before calling into module-specific method and decrement it after it returns.

remove_proc_entry() will wait until all readers and writers are done. To do this reliably it will set ->proc_fops to NULL and generic proc code won't call into module if it sees NULL ->proc_fops.

This patch implements part above. So far, no changes in modules code required. To proceed, let's look into ->proc_fops values during and after PDE creation:

```
pde = create_proc_entry();  
if (pde)  
    pde->proc_fops = ...;
```

proc_create() create empty PDE; (->proc_fops is NULL)
proc_register() glues PDE to /proc (->proc_fops is NULL)
proc_register() sets ->proc_fops to default (->proc_fops valid)
[module sets ->proc_fops to it's own (->proc_fops) valid]

Observation: ->proc_fops is not NULL, when create_proc_entry() exits.

Next set of races come into play:

```
pde = create_proc_entry();  
if (pde) {  
    pde->read_proc = ...;  
    pde->data = ...;  
}
```

Almost all ->read_proc, ->write_proc callbacks assume that ->data is valid when they're called. They cast ->data and dereference it.

To fix this we need a way to indicate that PDE is not readable and writeable.
->proc_fops nicely fits, because modules setting ->proc_fops only don't need changes at all. create_proc_entry() will exit with NULL ->proc_fops and helpers will be needed sometimes to set it.

1. Module sets ->proc_fops only
no changes
2. Module sets ->data and ->proc_fops
use a helper to set ->data before ->proc_fops and a barrier.
2. Module uses only create_proc_read_entry()
changes only in create_proc_read_entry();
3. Module uses only create_proc_info_entry()
changes only in create_proc_info_entry();
4. Module sets combination of ->data, ->read_proc, ->write_proc
use helper which will set fields, barrier and sets default ->proc_fops.
the best name I've come up so far is

```
void set_proc_entry_data_rw(struct proc_dir_entry *, void *, read_proc_t, write_proc_t);
```

Helper(s) will be introduced, then create_proc_entry() will start exiting with NULL ->proc_fops. After that use of helper(s) will become mandatory. Grepping for offenders will be easy (read_proc, ... are good names). ->data is bad name, however, after helpers will be plugged we can rename ->data to ->pde_data and it'll become good name.

Sorry, for somewhat chaotic explanations, please, comment on patch and RFC.

Signed-off-by: Alexey Dobriyan <adobriyan@openvz.org>

fs/proc/generic.c | 32 ++++++-----
include/linux/proc_fs.h | 2 ++
2 files changed, 30 insertions(+), 4 deletions(-)

```
--- a/fs/proc/generic.c
+++ b/fs/proc/generic.c
@@ -19,6 +19,7 @@ #include <linux/init.h>
#include <linux/idr.h>
#include <linux/namei.h>
#include <linux/bitops.h>
+#include <linux/delay.h>
#include <linux/spinlock.h>
#include <asm/uaccess.h>

@@ -76,6 +77,12 @@ proc_file_read(struct file *file, char _
if (!(page = (char*) __get_free_page(GFP_KERNEL)))
return -ENOMEM;

+ if (!dp->proc_fops)
+ goto out_free;
+ atomic_inc(&dp->pde_users);
+ if (!dp->proc_fops)
+ goto out_dec;
+
while ((nbytes > 0) && !eof) {
count = min_t(size_t, PROC_BLOCK_SIZE, nbytes);

@@ -195,6 +202,9 @@ proc_file_read(struct file *file, char _
buf += n;
retval += n;
}
+out_dec:
+ atomic_dec(&dp->pde_users);
+out_free:
free_page((unsigned long) page);
return retval;
}
@@ -205,14 +215,20 @@ proc_file_write(struct file *file, const
{
struct inode *inode = file->f_path.dentry->d_inode;
struct proc_dir_entry * dp;
+ ssize_t rv;

dp = PDE(inode);

- if (!dp->write_proc)
+ if (!dp->write_proc || !dp->proc_fops)
```

```

return -EIO;

- /* FIXME: does this routine need ppos? probably... */
- return dp->write_proc(file, buffer, count, dp->data);
+ rv = -EIO;
+ atomic_inc(&dp->pde_users);
+ if (dp->proc_fops)
+ /* FIXME: does this routine need ppos? probably... */
+ rv = dp->write_proc(file, buffer, count, dp->data);
+ atomic_dec(&dp->pde_users);
+ return rv;
}

@@ -717,12 +733,20 @@ void remove_proc_entry(const char *name,
    if (!parent && xlate_proc_name(name, &parent, &fn) != 0)
        goto out;
    len = strlen(fn);
-
+again:
    spin_lock(&proc_subdir_lock);
    for (p = &parent->subdir; *p; p=&(*p)->next ) {
        if (!proc_match(len, fn, *p))
            continue;
        de = *p;
+
+ de->proc_fops = NULL;
+ if (atomic_read(&de->pde_users) > 0) {
+     spin_unlock(&proc_subdir_lock);
+     msleep(1);
+     goto again;
+ }
+
+ *p = de->next;
+ de->next = NULL;
+ if (S_ISDIR(de->mode))
--- a/include/linux/proc_fs.h
+++ b/include/linux/proc_fs.h
@@ -66,6 +66,8 @@ struct proc_dir_entry {
    atomic_t count; /* use count */
    int deleted; /* delete flag */
    void *set;
+ atomic_t pde_users; /* number of readers + number of writers via
+  * ->read_proc, ->write_proc, ->get_info */
};

struct kcore_list {

```
