Subject: Racy /proc creations interfaces
Posted by adobriyan on Wed, 27 Dec 2006 13:36:04 GMT
View Forum Message <> Reply to Message

1. Not setting ->owner for your favourite /proc file is oopsable.

```
 $ while true; do cat /proc/fs/xfs/stat 1>/dev/null 2>/dev/null; done
 # while true; do modprobe xfs; rmmod xfs; done
```

```
BUG: unable to handle kernel paging request at virtual address fc281049
 printing eip:
fc281049
*pde = 37ca7067
Oops: 0000 [#1]
Modules linked in: af_packet ohci_hcd e1000 ehci_hcd uhci_hcd usbcore
CPU:    0
EIP:    0060:[<fc281049>]    Not tainted VLI
EFLAGS: 00010282   (2.6.19 #2)
EIP is at 0xfc281049
eax: f0705000   ebx: 00000000   ecx: 00000105   edx: f1e5ff68
esi: f0705000   edi: fc281049   ebp: 00000c00   esp: f1e5ff4c
ds: 007b   es: 007b   ss: 0068
Process cat (pid: 32588, ti=f1e5e000 task=f4377030 task.ti=f1e5e000)
Stack: c0177d9c 00000c00 f1e5ff6c 00000000 00001000 08051000 f7ccc240 00000000
       00000000 edbe82c0 00001000 08051000 f1e5ffa4 c0149f05 f1e5ffa4 c0177be5
       edbe82c0 fffffff7 08051000 f1e5e000 c014a26d f1e5ffa4 00000105 00000000
Call Trace:
 [<c0177d9c>] proc_file_read+0x1b7/0x262
 [<c0149f05>] vfs_read+0x85/0xf3
 [<c0177be5>] proc_file_read+0x0/0x262
 [<c014a26d>] sys_read+0x41/0x6a
 [<c0102be1>] sysenter_past_esp+0x56/0x79
 =======================
Code:  Bad EIP value.
EIP: [<fc281049>] 0xfc281049 SS:ESP 0068:f1e5ff4c
```

2. After adding ->owner for your favourite /proc file it's still looks racy

```
 pde = create_proc_entry("foo", 0644, NULL);
 /* so what? pde->owner is NULL here */
 if (pde)
  pde->owner = THIS_MODULE;
```

3. It's more: setting both ->read_proc and ->data after proc entry creation
   should be oopsable too, if ->read_proc operates on data in a non-trivial
   way (or ->write_proc).

sound/oss/trident.c:

```
 4446    res = create_proc_entry("ALi5451", 0, NULL);
 4447    if (res) {
 4448     res->write_proc = ali_write_proc;
      /* ->data is NULL here. */
 4449     res->data = card;
 4450    }
```

ali_write_proc() does spin_lock_irqsave on private card lock which is believed to be valid.

4. Proposed semantics: PDE should be in maximally working state before gluing into main /proc tree (proc_register()).

```
   struct proc_entry_raw foo_pe_raw = {
 .owner = THIS_MODULE,
 .name = "foo",
 .mode = 0644,
 .read_proc = foo_read_proc,
 .data = foo_data,
 .parent = foo_parent,
};

pde = create_proc_entry(&foo_pe_raw);
if (!pde)
 return -ENOMEM;
```

where "struct proc_entry_raw" is cut down version of "struct proc_dir_entry" where new create_proc_entry() would do

```
   proc_create() or kmalloc()
fill in fields from passed "struct proc_entry_raw"
proc_register()
```

Comments?

Below is 0-order approximation to what I mean (not tested at all):

```
--- a/fs/afs/proc.c
+++ b/fs/afs/proc.c
@@ -146,17 +146,23 @@ int afs_proc_init(void)
   goto error;
  proc_afs->owner = THIS_MODULE;

- p = create_proc_entry("cells", 0, proc_afs);
+ p = __create_proc_entry(&(struct proc_entry_raw){
+   .name = "cells",
+   .parent = proc_afs,
+   .proc_fops = &afs_proc_cells_fops,
```

```
+   .owner = THIS_MODULE,
+  });
  if (!p)
   goto error_proc;
- p->proc_fops = &afs_proc_cells_fops;
- p->owner = THIS_MODULE;

- p = create_proc_entry("rootcell", 0, proc_afs);
+ p = __create_proc_entry(&(struct proc_entry_raw){
+   .name = "rootcell",
+   .parent = proc_afs,
+   .proc_fops = &afs_proc_rootcell_fops,
+   .owner = THIS_MODULE,
+  });
  if (!p)
   goto error_cells;
- p->proc_fops = &afs_proc_rootcell_fops;
- p->owner = THIS_MODULE;

  _leave(" = 0");
  return 0;
@@ -429,26 +435,35 @@ int afs_proc_cell_setup(struct afs_cell
  if (!cell->proc_dir)
   return -ENOMEM;

- p = create_proc_entry("servers", 0, cell->proc_dir);
+ p = __create_proc_entry(&(struct proc_entry_raw){
+   .name = "servers",
+   .parent = cell->proc_dir,
+   .proc_fops = &afs_proc_cell_servers_fops,
+   .owner = THIS_MODULE,
+   .data = cell,
+  });
  if (!p)
   goto error_proc;
- p->proc_fops = &afs_proc_cell_servers_fops;
- p->owner = THIS_MODULE;
- p->data = cell;

- p = create_proc_entry("vlservers", 0, cell->proc_dir);
+ p = __create_proc_entry(&(struct proc_entry_raw){
+   .name = "vlservers",
+   .parent = cell->proc_dir,
+   .proc_fops = &afs_proc_cell_vlservers_fops,
+   .owner = THIS_MODULE,
+   .data = cell,
+  });
  if (!p)
```

```
   goto error_servers;
- p->proc_fops = &afs_proc_cell_vlservers_fops;
- p->owner = THIS_MODULE;
- p->data = cell;

- p = create_proc_entry("volumes", 0, cell->proc_dir);
+ p = __create_proc_entry(&(struct proc_entry_raw){
+   .name = "volumes",
+   .parent = cell->proc_dir,
+   .proc_fops = &afs_proc_cell_volumes_fops,
+   .owner = THIS_MODULE,
+   .data = cell,
+  });
  if (!p)
   goto error_vlservers;
- p->proc_fops = &afs_proc_cell_volumes_fops;
- p->owner = THIS_MODULE;
- p->data = cell;

  _leave(" = 0");
  return 0;
diff --git a/fs/jbd/journal.c b/fs/jbd/journal.c
index 10fff94..faf0470 100644
--- a/fs/jbd/journal.c
+++ b/fs/jbd/journal.c
@@ -1975,12 +1975,12 @@ #define JBD_PROC_NAME "sys/fs/jbd-debug"

 static void __init create_jbd_proc_entry(void)
 {
- proc_jbd_debug = create_proc_entry(JBD_PROC_NAME, 0644, NULL);
- if (proc_jbd_debug) {
-  /* Why is this so hard? */
-  proc_jbd_debug->read_proc = read_jbd_debug;
-  proc_jbd_debug->write_proc = write_jbd_debug;
- }
+ __create_proc_entry(&(struct proc_entry_raw){
+   .name = JBD_PROC_NAME,
+   .mode = 0644,
+   .read_proc = read_jbd_debug,
+   .write_proc = write_jbd_debug,
+  });
 }

 static void __exit remove_jbd_proc_entry(void)
diff --git a/fs/jbd2/journal.c b/fs/jbd2/journal.c
index 44fc32b..32de473 100644
--- a/fs/jbd2/journal.c
+++ b/fs/jbd2/journal.c
```

```
@@ -1986,12 +1986,12 @@ #define JBD_PROC_NAME "sys/fs/jbd2-debug

 static void __init create_jbd_proc_entry(void)
 {
- proc_jbd_debug = create_proc_entry(JBD_PROC_NAME, 0644, NULL);
- if (proc_jbd_debug) {
-  /* Why is this so hard? */
-  proc_jbd_debug->read_proc = read_jbd_debug;
-  proc_jbd_debug->write_proc = write_jbd_debug;
- }
+ __create_proc_entry(&(struct proc_entry_raw){
+    .name = JBD_PROC_NAME,
+    .mode = 0644,
+    .read_proc = read_jbd_debug,
+    .write_proc = write_jbd_debug,
+  });
 }

 static void __exit jbd2_remove_jbd_proc_entry(void)
diff --git a/fs/jffs/jffs_proc.c b/fs/jffs/jffs_proc.c
index 9bdd99a..ec00942 100644
--- a/fs/jffs/jffs_proc.c
+++ b/fs/jffs/jffs_proc.c
@@ -85,12 +85,12 @@ int jffs_register_jffs_proc_dir(int mtd,
  if (!part_root)
   goto out1;

- /* Create entry for 'info' file */
- part_info = create_proc_entry ("info", 0, part_root);
- if (!part_info)
+ if (!__create_proc_entry(&(struct proc_entry_raw){
+    .name = "info",
+    .parent = part_root,
+    .read_proc = jffs_proc_info_read,
+    .data = (void *)c,}))
   goto out2;
- part_info->read_proc = jffs_proc_info_read;
- part_info->data = (void *) c;

  /* Create entry for 'layout' file */
  part_layout = create_proc_entry ("layout", 0, part_root);
diff --git a/fs/nfs/client.c b/fs/nfs/client.c
index 23ab145..44ca278 100644
--- a/fs/nfs/client.c
+++ b/fs/nfs/client.c
@@ -1406,20 +1406,27 @@ int __init nfs_fs_proc_init(void)
  proc_fs_nfs->owner = THIS_MODULE;
```

```
  /* a file of servers with which we're dealing */
- p = create_proc_entry("servers", S_IFREG|S_IRUGO, proc_fs_nfs);
+ p = __create_proc_entry(&(struct proc_entry_raw){
+    .name = "servers",
+    .mode = S_IFREG|S_IRUGO,
+    .parent = proc_fs_nfs,
+    .proc_fops = &nfs_server_list_fops,
+    .owner = THIS_MODULE,
+  });
  if (!p)
   goto error_1;

- p->proc_fops = &nfs_server_list_fops;
- p->owner = THIS_MODULE;
-
  /* a file of volumes that we have mounted */
- p = create_proc_entry("volumes", S_IFREG|S_IRUGO, proc_fs_nfs);
+ p = __create_proc_entry(&(struct proc_entry_raw){
+    .name = "volumes",
+    .mode = S_IFREG|S_IRUGO,
+    .parent = proc_fs_nfs,
+    .proc_fops = &nfs_volume_list_fops,
+    .owner = THIS_MODULE,
+  });
  if (!p)
   goto error_2;

- p->proc_fops = &nfs_volume_list_fops;
- p->owner = THIS_MODULE;
  return 0;

 error_2:
diff --git a/fs/proc/generic.c b/fs/proc/generic.c
index 853cb87..0bce166 100644
--- a/fs/proc/generic.c
+++ b/fs/proc/generic.c
@@ -657,6 +657,47 @@ struct proc_dir_entry *proc_mkdir(const
  return proc_mkdir_mode(name, S_IRUGO | S_IXUGO, parent);
 }

+struct proc_dir_entry *__create_proc_entry(struct proc_entry_raw *raw)
+{
+ struct proc_dir_entry *ent;
+ mode_t mode = raw->mode;
+ nlink_t nlink;
+
+ if (S_ISDIR(mode)) {
+  if ((mode & S_IALLUGO) == 0)
```

```
+   mode |= S_IRUGO | S_IXUGO;
+  nlink = 2;
+ } else {
+  if ((mode & S_IFMT) == 0)
+   mode |= S_IFREG;
+  if ((mode & S_IALLUGO) == 0)
+   mode |= S_IRUGO;
+  nlink = 1;
+ }
+
+ ent = proc_create(&raw->parent, raw->name, mode, nlink);
+ if (ent) {
+  if (S_ISDIR(mode)) {
+   ent->proc_fops = &proc_dir_operations;
+   ent->proc_iops = &proc_dir_inode_operations;
+  }
+
+  ent->size = raw->size;
+  if (raw->proc_fops)
+   ent->proc_fops = raw->proc_fops;
+  ent->data = raw->data;
+  ent->read_proc = raw->read_proc;
+  ent->write_proc = raw->write_proc;
+  ent->owner = raw->owner;
+
+  if (proc_register(raw->parent, ent) < 0) {
+   kfree(ent);
+   ent = NULL;
+  }
+ }
+ return ent;
+}
+
 struct proc_dir_entry *create_proc_entry(const char *name, mode_t mode,
       struct proc_dir_entry *parent)
 {
diff --git a/fs/proc/root.c b/fs/proc/root.c
index 64d242b..f7d3fda 100644
--- a/fs/proc/root.c
+++ b/fs/proc/root.c
@@ -166,6 +166,7 @@ struct proc_dir_entry proc_root = {
 EXPORT_SYMBOL(proc_symlink);
 EXPORT_SYMBOL(proc_mkdir);
 EXPORT_SYMBOL(create_proc_entry);
+EXPORT_SYMBOL(__create_proc_entry);
 EXPORT_SYMBOL(remove_proc_entry);
 EXPORT_SYMBOL(proc_root);
 EXPORT_SYMBOL(proc_root_fs);
```

```
diff --git a/include/linux/proc_fs.h b/include/linux/proc_fs.h
index 87dec8f..0523180 100644
--- a/include/linux/proc_fs.h
+++ b/include/linux/proc_fs.h
@@ -68,6 +68,18 @@ struct proc_dir_entry {
  void *set;
 };

+struct proc_entry_raw {
+ const char *name;
+ mode_t mode;
+ loff_t size;
+ const struct file_operations *proc_fops;
+ struct module *owner;
+ struct proc_dir_entry *parent;
+ void *data;
+ read_proc_t *read_proc;
+ write_proc_t *write_proc;
+};
+
 struct kcore_list {
  struct kcore_list *next;
  unsigned long addr;
@@ -107,6 +119,7 @@ char *task_mem(struct mm_struct *, char

 extern struct proc_dir_entry *create_proc_entry(const char *name, mode_t mode,
       struct proc_dir_entry *parent);
+struct proc_dir_entry *__create_proc_entry(struct proc_entry_raw *raw);
 extern void remove_proc_entry(const char *name, struct proc_dir_entry *parent);

 extern struct vfsmount *proc_mnt;
@@ -214,6 +227,11 @@ static inline void proc_flush_task(struc
 static inline struct proc_dir_entry *create_proc_entry(const char *name,
  mode_t mode, struct proc_dir_entry *parent) { return NULL; }

+static inline struct proc_dir_entry *__create_proc_entry(struct proc_entry_raw *raw)
+{
+ return NULL;
+}
+
 #define remove_proc_entry(name, parent) do {} while (0)

 static inline struct proc_dir_entry *proc_symlink(const char *name,
diff --git a/kernel/configs.c b/kernel/configs.c
index 8fa1fb2..1694608 100644
--- a/kernel/configs.c
+++ b/kernel/configs.c
@@ -85,18 +85,16 @@ static const struct file_operations ikco
```

```
 static int __init ikconfig_init(void)
 {
- struct proc_dir_entry *entry;
+ struct proc_entry_raw pe_raw = {
+  .name  = "config.gz",
+  .mode  = S_IFREG | S_IRUGO,
+  .parent  = &proc_root,
+  .proc_fops = &ikconfig_file_ops,
+  .size  = kernel_config_data_size,
+ };

  /* create the current config file */
- entry = create_proc_entry("config.gz", S_IFREG | S_IRUGO,
-       &proc_root);
- if (!entry)
-  return -ENOMEM;
-
- entry->proc_fops = &ikconfig_file_ops;
- entry->size = kernel_config_data_size;
-
- return 0;
+ return __create_proc_entry(&pe_raw) ? 0 : -ENOMEM;
 }

 /*****************************************************/
diff --git a/kernel/dma.c b/kernel/dma.c
index 937b13c..51726c0 100644
--- a/kernel/dma.c
+++ b/kernel/dma.c
@@ -149,12 +149,11 @@ static const struct file_operations proc

 static int __init proc_dma_init(void)
 {
- struct proc_dir_entry *e;
-
- e = create_proc_entry("dma", 0, NULL);
- if (e)
-  e->proc_fops = &proc_dma_operations;
-
+ struct proc_entry_raw pe_raw = {
+  .name    = "dma",
+  .proc_fops = &proc_dma_operations,
+ };
+ __create_proc_entry(&pe_raw);
  return 0;
 }
```

```
diff --git a/kernel/irq/proc.c b/kernel/irq/proc.c
index 61f5c71..aecabf5 100644
--- a/kernel/irq/proc.c
+++ b/kernel/irq/proc.c
@@ -130,17 +130,17 @@ void register_irq_proc(unsigned int irq)

 #ifdef CONFIG_SMP
  {
-  struct proc_dir_entry *entry;
+  struct proc_entry_raw pe_raw = {
+   .name = "smp_affinity",
+   .mode = 0600,
+   .parent = irq_desc[irq].dir,
+   .data = (void *)(long)irq,
+   .read_proc = irq_affinity_read_proc,
+   .write_proc = irq_affinity_write_proc,
+  };

    /* create /proc/irq/<irq>/smp_affinity */
-  entry = create_proc_entry("smp_affinity", 0600, irq_desc[irq].dir);
-
-  if (entry) {
-   entry->nlink = 1;
-   entry->data = (void *)(long)irq;
-   entry->read_proc = irq_affinity_read_proc;
-   entry->write_proc = irq_affinity_write_proc;
-  }
+  __create_proc_entry(&pe_raw);
  }
 #endif
 }
diff --git a/kernel/kallsyms.c b/kernel/kallsyms.c
index 6f294ff..3f26794 100644
--- a/kernel/kallsyms.c
+++ b/kernel/kallsyms.c
@@ -442,11 +442,12 @@ static const struct file_operations kall

 static int __init kallsyms_init(void)
 {
- struct proc_dir_entry *entry;
-
- entry = create_proc_entry("kallsyms", 0444, NULL);
- if (entry)
-  entry->proc_fops = &kallsyms_operations;
+ struct proc_entry_raw pe_raw = {
+  .name    = "kallsyms",
+  .mode    = 0444,
+  .proc_fops = &kallsyms_operations,
```

```
+ };
+ __create_proc_entry(&pe_raw);
  return 0;
 }
 __initcall(kallsyms_init);
diff --git a/kernel/lockdep_proc.c b/kernel/lockdep_proc.c
index b554b40..3d297d4 100644
--- a/kernel/lockdep_proc.c
+++ b/kernel/lockdep_proc.c
@@ -328,16 +328,19 @@ static const struct file_operations proc

 static int __init lockdep_proc_init(void)
 {
- struct proc_dir_entry *entry;
-
- entry = create_proc_entry("lockdep", S_IRUSR, NULL);
- if (entry)
-  entry->proc_fops = &proc_lockdep_operations;
-
- entry = create_proc_entry("lockdep_stats", S_IRUSR, NULL);
- if (entry)
-  entry->proc_fops = &proc_lockdep_stats_operations;
-
+ struct proc_entry_raw pe_raw_lockdep = {
+  .name = "lockdep",
+  .mode = S_IRUSR,
+  .proc_fops = &proc_lockdep_operations,
+ };
+ struct proc_entry_raw pe_raw_lockdep_stats = {
+  .name = "lockdep_stats",
+  .mode = S_IRUSR,
+  .proc_fops = &proc_lockdep_stats_operations,
+ };
+
+ __create_proc_entry(&pe_raw_lockdep);
+ __create_proc_entry(&pe_raw_lockdep_stats);
  return 0;
 }

diff --git a/kernel/profile.c b/kernel/profile.c
index fb5e03d..19b73a0 100644
--- a/kernel/profile.c
+++ b/kernel/profile.c
@@ -429,15 +429,16 @@ static int prof_cpu_mask_write_proc (str

 void create_prof_cpu_mask(struct proc_dir_entry *root_irq_dir)
 {
- struct proc_dir_entry *entry;
```

```
-
+ struct proc_entry_raw pe_raw = {
+  .name = "prof_cpu_mask",
+  .mode = 0600,
+  .parent = root_irq_dir,
+  .data = &prof_cpu_mask,
+  .read_proc = prof_cpu_mask_read_proc,
+  .write_proc = prof_cpu_mask_write_proc,
+ };
   /* create /proc/irq/prof_cpu_mask */
- if (!(entry = create_proc_entry("prof_cpu_mask", 0600, root_irq_dir)))
-  return;
- entry->nlink = 1;
- entry->data = (void *)&prof_cpu_mask;
- entry->read_proc = prof_cpu_mask_read_proc;
- entry->write_proc = prof_cpu_mask_write_proc;
+ __create_proc_entry(&pe_raw);
 }

 /*
@@ -561,16 +562,19 @@ #endif

 static int __init create_proc_profile(void)
 {
- struct proc_dir_entry *entry;
+ struct proc_entry_raw pe_raw = {
+  .name    = "profile",
+  .mode    = S_IWUSR | S_IRUGO,
+  .proc_fops = &proc_profile_operations,
+  .size    = (1 + prof_len) * sizeof(atomic_t),
+ };

   if (!prof_on)
    return 0;
   if (create_hash_tables())
    return -1;
- if (!(entry = create_proc_entry("profile", S_IWUSR | S_IRUGO, NULL)))
+ if (!__create_proc_entry(&pe_raw))
    return 0;
- entry->proc_fops = &proc_profile_operations;
- entry->size = (1+prof_len) * sizeof(atomic_t);
   hotcpu_notifier(profile_cpu_callback, 0);
   return 0;
 }
diff --git a/kernel/resource.c b/kernel/resource.c
index 7b9a497..7caf831 100644
--- a/kernel/resource.c
+++ b/kernel/resource.c
```

```
@@ -131,14 +131,17 @@ static const struct file_operations proc

 static int __init ioresources_init(void)
 {
- struct proc_dir_entry *entry;
-
- entry = create_proc_entry("ioports", 0, NULL);
- if (entry)
-  entry->proc_fops = &proc_ioports_operations;
- entry = create_proc_entry("iomem", 0, NULL);
- if (entry)
-  entry->proc_fops = &proc_iomem_operations;
+ struct proc_entry_raw pe_raw_ioports = {
+  .name = "ioports",
+  .proc_fops = &proc_ioports_operations,
+ };
+ struct proc_entry_raw pe_raw_iomem = {
+  .name = "iomem",
+  .proc_fops = &proc_iomem_operations,
+ };
+
+ __create_proc_entry(&pe_raw_ioports);
+ __create_proc_entry(&pe_raw_iomem);
  return 0;
 }
 __initcall(ioresources_init);
```

Subject: Re: Racy /proc creations interfaces
Posted by Al Viro on Wed, 27 Dec 2006 13:56:24 GMT

View Forum Message <> Reply to Message

On Wed, Dec 27, 2006 at 04:42:23PM +0300, Alexey Dobriyan wrote:
>
>   struct proc_entry_raw foo_pe_raw = {
>   .owner = THIS_MODULE,
>   .name = "foo",
>   .mode = 0644,
>   .read_proc = foo_read_proc,
>   .data = foo_data,
>   .parent = foo_parent,
> };
>
> pde = create_proc_entry(&foo_pe_raw);
> if (!pde)
>   return -ENOMEM;
>
>   where "struct proc_entry_raw" is cut down version of "struct proc_dir_entry"

Ewwwwwwwwwwwwwww

Please, please no. Especially not .parent. If anything, let's add a
helper saying "it's all set up now". And turn create_proc_entry()
into a macro that would pass THIS_MODULE to underlying function and
call that helper, so that simple cases wouldn't have to bother at all.

Subject: Re: Racy /proc creations interfaces
Posted by adobriyan on Thu, 28 Dec 2006 08:15:37 GMT
View Forum Message <> Reply to Message

On Wed, Dec 27, 2006 at 01:56:24PM +0000, Al Viro wrote:
> On Wed, Dec 27, 2006 at 04:42:23PM +0300, Alexey Dobriyan wrote:
> >
> >   struct proc_entry_raw foo_pe_raw = {
> >   .owner = THIS_MODULE,
> >   .name = "foo",
> >   .mode = 0644,
> >   .read_proc = foo_read_proc,
> >   .data = foo_data,
> >   .parent = foo_parent,
> > };
> >
> > pde = create_proc_entry(&foo_pe_raw);
> > if (!pde)
> >   return -ENOMEM;
> >
> >   where "struct proc_entry_raw" is cut down version of "struct proc_dir_entry"
>
> Ewwwwwwwwwwwwwww
>
> Please, please no. Especially not .parent. If anything, let's add a
> helper saying "it's all set up now". And turn create_proc_entry()
> into a macro that would pass THIS_MODULE to underlying function and
> call that helper, so that simple cases wouldn't have to bother at all.

People are setting ->data after create_proc_entry():

drivers/zorro/proc.c:
```
  110 static int __init zorro_proc_attach_device(u_int slot)
  111 {
  112  struct proc_dir_entry *entry;
  113  char name[4];
  114
  115  sprintf(name, "%02x", slot);
  116  entry = create_proc_entry(name, 0, proc_bus_zorro_dir);
```

```
  117  if (!entry)
  118   return -ENOMEM;
  119  entry->proc_fops = &proc_bus_zorro_operations;
  120  entry->data = &zorro_autocon[slot];
  121  entry->size = sizeof(struct zorro_dev);
```

If create_proc_entry is a macro doing what you suggest (am I right?)

```
#define create_proc_entry(name, mode, parent)
({
 struct proc_dir_entry *pde;

 pde = __create_proc_entry(name, mode, parent, THIS_MODULE);
 if (pde)
  mark_proc_entry_ready(pde);
 pde;
})
```

there is still a problem because we want it to be equivalent to

```
pde = create_proc_entry(...);
if (!pde)
 return -ENOMEM;
pde->proc_fops = ...;
pde->data = ...;
mark_proc_entry_ready(pde);
```

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 it it sees NULL ->proc_fops.

This patch implements part above. So far, no changes in modules code
required. To proceed, lets 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 {
```

Subject: Re: [PATCH] Fix rmmod/read/write races in /proc entries
Posted by Andrew Morton on Tue, 23 Jan 2007 20:58:01 GMT
View Forum Message <> Reply to Message

On Mon, 15 Jan 2007 18:39:27 +0300
Alexey Dobriyan <adobriyan@openvz.org> wrote:

```
> ----------------------------------------------------------
> 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 it it sees NULL ->proc_fops.
>
> This patch implements part above. So far, no changes in modules code
> required. To proceed, lets 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.

<head spins>

Looks a bit hacky. Can this race not be fixed by addition of suitable
locking, or possibly refcounting-under-locking?

> 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;
> +

You'll be shocked to know that I'd prefer more comments in there. Enough
for a later maintainer to be able to understand what's going on.

>   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;
> }

Here too.

>
> @@ -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;
> + }
> +

And here.

>   *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 {

Subject: Re: [PATCH] Fix rmmod/read/write races in /proc entries
Posted by adobriyan on Wed, 24 Jan 2007 15:16:41 GMT
View Forum Message <> Reply to Message

On Tue, Jan 23, 2007 at 12:58:01PM -0800, Andrew Morton wrote:
> <head spins>
>
> Looks a bit hacky. Can this race not be fixed by addition of suitable
> locking, or possibly refcounting-under-locking?

I'll think about it.

> > @@ -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;
> > +
>
> You'll be shocked to know that I'd prefer more comments in there. Enough
> for a later maintainer to be able to understand what's going on.

Here is replacement patch with rewritten changelog and comments in
place. HTH.

[PATCH] Fix rmmod/read/write races in /proc entries

Current /proc creation interfaces suffer from at least two types of races:
---
1. Write via ->write_proc sleeps in copy_from_user(). Module disappears
   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 */
   }
    *boom*
```
---
2. Read/write happens when PDE only partially initialized. ->data is NULL
   when create_proc_entry() returns. Almost all ->read_proc and
   ->write_proc handlers assume that ->data is valid.

```
   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 done
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 it it sees NULL ->proc_fops.

This patch implements part above. So far, no changes in proc users
required. Patch fixes races of type 1.

Unfortunately, fixing races of type #2 will require changing in some modules.

We need an indicator of PDE readinness of accepting reads and writes.
->proc_fops nicely fits. It is going to get new semantics:

* if ->proc_fops is valid, PDE will accept reads and writes via ->read_proc,
  ->write_proc, ->get_info.
* if ->proc_fops is NULL, PDE won't call into module's code.

remove_proc_entry() and only remove_proc_entry() will clear ->proc_fops.
create_proc_entry() will not set ->proc_fops. Helpers will be required.

Helpers will set ->proc_fops last (after ->data, particularly).

```
 set_proc_entry_data_fops(pde, data, fops);
 set_proc_entry_data_read_write(pde, data, read_proc, write_proc);
```

When all necessary helpers will be plugged, create_proc_entry will stop
setting default proc_fops and helpers will start setting it. Races of
type #2 will be fixed.

If module sets ->proc_fops only, or uses create_proc_read_entry(), or uses
create_proc_info_entry(), there won't be any changes for module.

```
 fs/proc/generic.c       |   61 ++++++++++++++++++++++++++++++++++++++++++++++---
 include/linux/proc_fs.h |   15 ++++++++++
 2 files changed, 73 insertions(+), 3 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 +77,25 @@ proc_file_read(struct file *file, char _
  if (!(page = (char*) __get_free_page(GFP_KERNEL)))
   return -ENOMEM;

+ if (!dp->proc_fops)
+  /*
+   * remove_proc_entry() marked PDE as "going away".
+   * No new readers allowed.
+   */
+  goto out_free;
+ /*
+  * We are going to call into module's code via ->get_info or
+  * ->read_proc. Bump refcount so that remove_proc_entry() will
+  * wait for read to complete.
+  */
+ atomic_inc(&dp->pde_users);
+ if (!dp->proc_fops)
+  /*
+   * While we're busy bumping refcount, remove_proc_entry()
+   * marked PDE as "going away". Obey.
+   */
+  goto out_dec;
+
```

```
  while ((nbytes > 0) && !eof) {
   count = min_t(size_t, PROC_BLOCK_SIZE, nbytes);

@@ -195,6 +215,9 @@ 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 +228,33 @@ 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)
    return -EIO;
+ /*
+  * remove_proc_entry() marked PDE as "going away".
+  * No new writers allowed.
+  */
+ if (!dp->proc_fops)
+  return -EIO;

- /* FIXME: does this routine need ppos?  probably... */
- return dp->write_proc(file, buffer, count, dp->data);
+ rv = -EIO;
+ /*
+  * We are going to call into module's code via ->write_proc .
+  * Bump refcount so that module won't dissapear while ->write_proc
+  * sleeps in copy_from_user(). remove_proc_entry() will wait for
+  * write to complete.
+  */
+ atomic_inc(&dp->pde_users);
+ if (dp->proc_fops)
+  /* PDE is ready, refcount bumped, call into module. */
+  /* 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 +759,25 @@ 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;
+
+  /*
+   * Stop accepting new readers/writers. If you're dynamically
+   * allocating ->proc_fops, save a pointer somewhere.
+   */
+  de->proc_fops = NULL;
+  /* Wait until all readers/writers are done. */
+  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
@@ -56,6 +56,19 @@ struct proc_dir_entry {
  gid_t gid;
  loff_t size;
  struct inode_operations * proc_iops;
+ /*
+  * NULL ->proc_fops means "PDE is going away RSN" or
+  * "PDE is just created". In either case ->get_info, ->read_proc,
+  * ->write_proc won't be called because it's too late or too early,
+  * respectively.
+  *
+  * Valid ->proc_fops means "use this file_operations" or
+  * "->data is setup, it's safe to call ->read_proc, ->write_proc which
+  * dereference it".
+  *
+  * If you're allocating ->proc_fops dynamically, save a pointer
+  * somewhere.
+  */
  const struct file_operations * proc_fops;
```

```
  get_info_t *get_info;
  struct module *owner;
@@ -66,6 +79,8 @@ 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 {
```