Subject: [PATCH RESEND 2/2] Fix some kallsyms_lookup() vs rmmod races
Posted by Alexey Dobriyan on Fri, 16 Mar 2007 11:39:01 GMT
View Forum Message <> Reply to Message

[cc'ing folks whose proc files are affected]

kallsyms_lookup() can call module_address_lookup() which iterates over
modules list without module_mutex taken. Comment at the top of
module_address_lookup() says it's for oops resolution so races are
irrelevant, but in some cases it's reachable from regular code:

BUG: unable to handle kernel paging request at virtual address f94fb19c
 printing eip:
c0138097
*pde = 33576067
Oops: 0000 [#1]
PREEMPT
Modules linked in: ohci_hcd af_packet e1000 ehci_hcd uhci_hcd usbcore
CPU: 0
EIP: 0060:[<c0138097>]   Not tainted VLI
EFLAGS: 00010246   (2.6.21-rc3-8b9909ded6922c33c221b105b26917780cfa497d #25)
EIP is at module_address_lookup+0x43/0x24c
eax: 00000000   ebx: 00000000   ecx: f94fb080   edx: f882e704
esi: 00000000   edi: 00000000   ebp: 00000000   esp: d30b8e78
ds: 007b   es: 007b   fs: 00d8 gs: 0033 ss: 0068
Process cat (pid: 13274, ti=d30b8000 task=d2b50af0 task.ti=d30b8000)
Stack: d30b8f44 d30b8f48 00008001 00000000 c039a23c d30b8ec4 00000000 d30b8f44
       d30b8f48 c013876c d30b8f4c 00000000 cf544000 ffffffff4 d38ebb7c c01880ff
       d30b8f4c d30b8ec4 00000246 c0399600 00000001 00000001 00000000 c0399654
Call Trace:
 [<c013876c>] kallsyms_lookup+0x5e/0x7c
 [<c01880ff>] proc_pid_wchan+0x38/0x76
 [<c01404fe>] __alloc_pages+0x4f/0x2f9
 [<c01891c6>] proc_info_read+0x6f/0xa8
 [<c015bbfd>] sys_fstat64+0x1e/0x23
 [<c01592d4>] vfs_read+0x7d/0xb5
 [<c0189157>] proc_info_read+0x0/0xa8
 [<c0159621>] sys_read+0x41/0x6a
 [<c0103e72>] sysenter_past_esp+0x5f/0x99
 =======================
Code: 04 8b 51 04 0f 18 02 90 81 3d 4c 8d 39 c0 4c 8d 39 c0 74 3f 8b b9 18 01 00 00 8b 99 10
01 00 00 39 eb 77 07 8d 04 3b 39 c5 72 32 <8b> b1 1c 01 00 00 8b 81 14 01 00 00 39 c5 72 06
01 f0 39 c5 72
EIP: [<c0138097>] module_address_lookup+0x43/0x24c SS:ESP 0068:d30b8e78

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

```
 fs/proc/base.c           |    7 +++++++
 kernel/time/timer_list.c |    6 ++++++
 kernel/time/timer_stats.c |    6 ++++++
 mm/slab.c                |    3 +++
 4 files changed, 22 insertions(+)

--- a/fs/proc/base.c
+++ b/fs/proc/base.c
@@ -61,6 +61,7 @@ #include <linux/seq_file.h>
 #include <linux/namei.h>
 #include <linux/mnt_namespace.h>
 #include <linux/mm.h>
+#include <linux/module.h>
 #include <linux/smp_lock.h>
 #include <linux/rcupdate.h>
 #include <linux/kallsyms.h>
@@ -282,7 +283,13 @@ static int proc_pid_wchan(struct task_st

  wchan = get_wchan(task);

+ module_mutex_lock();
  sym_name = kallsyms_lookup(wchan, &size, &offset, &modname, namebuf);
+ /*
+  * size, offset, modname aren't used, sym_name is copied into namebuf,
+  * so drop it now.
+  */
+ module_mutex_unlock();
  if (sym_name)
   return sprintf(buffer, "%s", sym_name);
  return sprintf(buffer, "%lu", wchan);
--- a/kernel/time/timer_list.c
+++ b/kernel/time/timer_list.c
@@ -44,7 +44,13 @@ static void print_name_offset(struct seq
  const char *sym_name;
  char *modname;

+ module_mutex_lock();
  sym_name = kallsyms_lookup(addr, &size, &offset, &modname, namebuf);
+ /*
+  * size, offset, modname aren't used, sym_name is copied into namebuf,
+  * so drop it now.
+  */
+ module_mutex_unlock();
  if (sym_name)
   SEQ_printf(m, "%s", sym_name);
  else
--- a/kernel/time/timer_stats.c
+++ b/kernel/time/timer_stats.c
```

```
@@ -262,7 +262,13 @@ static void print_name_offset(struct seq
  const char *sym_name;
  char *modname;

+ module_mutex_lock();
  sym_name = kallsyms_lookup(addr, &size, &offset, &modname, namebuf);
+ /*
+  * size, offset, modname aren't used, sym_name is copied into namebuf,
+  * so drop it now.
+  */
+ module_mutex_unlock();
  if (sym_name)
   seq_printf(m, "%s", sym_name);
  else
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -4385,14 +4385,17 @@ #ifdef CONFIG_KALLSYMS
  unsigned long offset, size;
  char namebuf[KSYM_NAME_LEN+1];

+ module_mutex_lock();
  name = kallsyms_lookup(address, &size, &offset, &modname, namebuf);

  if (name) {
   seq_printf(m, "%s+%#lx/%#lx", name, offset, size);
   if (modname)
    seq_printf(m, " [%s]", modname);
+  module_mutex_unlock();
   return;
  }
+ module_mutex_unlock();
 #endif
  seq_printf(m, "%p", (void *)address);
 }
```

Subject: Re: [PATCH RESEND 2/2] Fix some kallsyms_lookup() vs rmmod races
Posted by Ingo Molnar on Fri, 16 Mar 2007 11:51:54 GMT
View Forum Message <> Reply to Message

* Alexey Dobriyan <adobriyan@sw.ru> wrote:

> [cc'ing folks whose proc files are affected]
>
> kallsyms_lookup() can call module_address_lookup() which iterates over
> modules list without module_mutex taken. Comment at the top of
> module_address_lookup() says it's for oops resolution so races are
> irrelevant, but in some cases it's reachable from regular code:

looking at the problem from another angle: wouldnt this be something
that would benefit from freeze_processes()/unfreeze_processes(), and
hence no locking would be required?

Ingo

Subject: Re: [PATCH RESEND 2/2] Fix some kallsyms_lookup() vs rmmod races
Posted by Paulo Marques on Fri, 16 Mar 2007 16:16:10 GMT
View Forum Message <> Reply to Message

Ingo Molnar wrote:
> * Alexey Dobriyan <adobriyan@sw.ru> wrote:
>
>> [cc'ing folks whose proc files are affected]
>>
>> kallsyms_lookup() can call module_address_lookup() which iterates over
>> modules list without module_mutex taken. Comment at the top of
>> module_address_lookup() says it's for oops resolution so races are
>> irrelevant, but in some cases it's reachable from regular code:
>
> looking at the problem from another angle: wouldnt this be something
> that would benefit from freeze_processes()/unfreeze_processes(), and
> hence no locking would be required?

I also considered this, but it seemed a little too "blunt" to stop
everything (including completely unrelated processes and kernel threads)
just to remove a module.

How about something like this instead? (just for review)

It's a little more intrusive, since the interface for symbol lookup is
changed (and it affects the callers), but it makes the caller aware that
it should set "safe_to_lock" if possible.

This way we avoid exposing module_mutex outside of module.c and avoid
returning any data pointer to some data structure that might disappear
before the caller tries to use it.

Some of these changes are actually cleanups, because the callers where
creating a dummy modname variables that they didn't use afterwards.

The thing I like the less about this patch is the increase of stack
usage on some code paths by about 60 bytes.

Anyway, I don't have very strong feelings about this, so if you think it
would be better to use freeze_processes()/unfreeze_processes(), I can

cook up a patch for that too...

--
Paulo Marques - www.grupopie.com

"Very funny Scotty. Now beam up my clothes."

diffstat:
> arch/parisc/kernel/unwind.c | 3 --
> arch/powerpc/xmon/xmon.c | 11 ++++-----
> arch/ppc/xmon/xmon.c | 8 +++---
> arch/sh64/kernel/unwind.c | 4 +--
> arch/x86_64/kernel/traps.c | 10 ++++----
> fs/proc/base.c | 3 --
> include/linux/kallsyms.h | 6 +++-
> include/linux/module.h | 44 ++++++++++++++++++++----------
> kernel/kallsyms.c | 53 ++++++++++++++++++++++++++----------
> kernel/kprobes.c | 6 ++--
> kernel/lockdep.c | 3 --
> kernel/module.c | 44 ++++++++++++++++++++++++++---------
> kernel/time/timer_list.c | 3 --
> kernel/time/timer_stats.c | 3 --
> mm/slab.c | 6 ++--
> 15 files changed, 114 insertions(+), 93 deletions(-)

```
diff --git a/arch/parisc/kernel/unwind.c b/arch/parisc/kernel/unwind.c
--- a/arch/parisc/kernel/unwind.c
+++ b/arch/parisc/kernel/unwind.c
@@ -216,11 +216,10 @@ static void unwind_frame_regs(struct unw
   /* Handle some frequent special cases.... */
   {
    char symname[KSYM_NAME_LEN+1];
-   char *modname;
    unsigned long symsize, offset;

    kallsyms_lookup(info->ip, &symsize, &offset,
-     &modname, symname);
+     NULL, symname, 0);

    dbg("info->ip = 0x%lx, name = %s\n", info->ip, symname);

diff --git a/arch/powerpc/xmon/xmon.c b/arch/powerpc/xmon/xmon.c
--- a/arch/powerpc/xmon/xmon.c
+++ b/arch/powerpc/xmon/xmon.c
@@ -1218,7 +1218,6 @@ static void get_function_bounds(unsigned
 {
  unsigned long size, offset;
```

```
  const char *name;
- char *modname;

  *startp = *endp = 0;
  if (pc == 0)
@@ -1226,7 +1225,7 @@ static void get_function_bounds(unsigned
  if (setjmp(bus_error_jmp) == 0) {
   catch_memory_errors = 1;
   sync();
-  name = kallsyms_lookup(pc, &size, &offset, &modname, tmpstr);
+  name = kallsyms_lookup(pc, &size, &offset, NULL, tmpstr, 0);
   if (name != NULL) {
    *startp = pc - offset;
    *endp = pc - offset + size;
@@ -2496,7 +2495,7 @@ symbol_lookup(void)
 static void xmon_print_symbol(unsigned long address, const char *mid,
         const char *after)
 {
- char *modname;
+ char modname[MODULE_NAME_LEN + 1];
  const char *name = NULL;
  unsigned long offset, size;

@@ -2504,8 +2503,8 @@ static void xmon_print_symbol(unsigned l
  if (setjmp(bus_error_jmp) == 0) {
   catch_memory_errors = 1;
   sync();
-  name = kallsyms_lookup(address, &size, &offset, &modname,
-           tmpstr);
+  name = kallsyms_lookup(address, &size, &offset, modname,
+           tmpstr, 0);
   sync();
   /* wait a little while to see if we get a machine check */
   __delay(200);
@@ -2515,7 +2514,7 @@ static void xmon_print_symbol(unsigned l

  if (name) {
   printf("%s%s+%#lx/%#lx", mid, name, offset, size);
-  if (modname)
+  if (modname[0])
    printf(" [%s]", modname);
  }
  printf("%s", after);
diff --git a/arch/ppc/xmon/xmon.c b/arch/ppc/xmon/xmon.c
--- a/arch/ppc/xmon/xmon.c
+++ b/arch/ppc/xmon/xmon.c
@@ -177,7 +177,7 @@ extern inline void __delay(unsigned int
 static void xmon_print_symbol(unsigned long address, const char *mid,
```

```
         const char *after)
 {
- char *modname;
+ char modname[MODULE_NAME_LEN+1];
  const char *name = NULL;
  unsigned long offset, size;
  static char tmpstr[128];
@@ -186,8 +186,8 @@ static void xmon_print_symbol(unsigned l
  if (setjmp(bus_error_jmp) == 0) {
   debugger_fault_handler = handle_fault;
   sync();
-  name = kallsyms_lookup(address, &size, &offset, &modname,
-           tmpstr);
+  name = kallsyms_lookup(address, &size, &offset, modname,
+           tmpstr, 0);
   sync();
   /* wait a little while to see if we get a machine check */
   __delay(200);
@@ -196,7 +196,7 @@ static void xmon_print_symbol(unsigned l

  if (name) {
   printf("%s%s+%#lx/%#lx", mid, name, offset, size);
-  if (modname)
+  if (modname[0])
    printf(" [%s]", modname);
  }
  printf("%s", after);
diff --git a/arch/sh64/kernel/unwind.c b/arch/sh64/kernel/unwind.c
--- a/arch/sh64/kernel/unwind.c
+++ b/arch/sh64/kernel/unwind.c
@@ -46,7 +46,7 @@ static int lookup_prev_stack_frame(unsig
         struct pt_regs *regs)
 {
  const char *sym;
- char *modname, namebuf[128];
+ char namebuf[128];
  unsigned long offset, size;
  unsigned long prologue = 0;
  unsigned long fp_displacement = 0;
@@ -54,7 +54,7 @@ static int lookup_prev_stack_frame(unsig
  unsigned long offset_r14 = 0, offset_r18 = 0;
  int i, found_prologue_end = 0;

- sym = kallsyms_lookup(pc, &size, &offset, &modname, namebuf);
+ sym = kallsyms_lookup(pc, &size, &offset, NULL, namebuf, 0);
  if (!sym)
   return -EINVAL;
```

```
diff --git a/arch/x86_64/kernel/traps.c b/arch/x86_64/kernel/traps.c
--- a/arch/x86_64/kernel/traps.c
+++ b/arch/x86_64/kernel/traps.c
@@ -116,18 +116,18 @@ void printk_address(unsigned long addres
 {
  unsigned long offset = 0, symsize;
  const char *symname;
- char *modname;
+ char modname[MODULE_NAME_LEN+1];
  char *delim = ":";
- char namebuf[128];
+ char namebuf[KSYM_NAME_LEN+1];

  symname = kallsyms_lookup(address, &symsize, &offset,
-     &modname, namebuf);
+     modname, namebuf, 0);
  if (!symname) {
   printk(" [<%016lx>]\n", address);
   return;
  }
- if (!modname)
-  modname = delim = "";
+ if (!modname[0])
+  delim = "";
  printk(" [<%016lx>] %s%s%s%s+0x%lx/0x%lx\n",
   address, delim, modname, delim, symname, offset, symsize);
 }
diff --git a/fs/proc/base.c b/fs/proc/base.c
--- a/fs/proc/base.c
+++ b/fs/proc/base.c
@@ -275,14 +275,13 @@ static int proc_pid_auxv(struct task_str
  */
 static int proc_pid_wchan(struct task_struct *task, char *buffer)
 {
- char *modname;
  const char *sym_name;
  unsigned long wchan, size, offset;
  char namebuf[KSYM_NAME_LEN+1];

  wchan = get_wchan(task);

- sym_name = kallsyms_lookup(wchan, &size, &offset, &modname, namebuf);
+ sym_name = kallsyms_lookup(wchan, &size, &offset, NULL, namebuf, 1);
  if (sym_name)
   return sprintf(buffer, "%s", sym_name);
  return sprintf(buffer, "%lu", wchan);
diff --git a/include/linux/kallsyms.h b/include/linux/kallsyms.h
--- a/include/linux/kallsyms.h
```

```
+++ b/include/linux/kallsyms.h
@@ -20,7 +20,8 @@ extern int kallsyms_lookup_size_offset(u
 const char *kallsyms_lookup(unsigned long addr,
        unsigned long *symbolsize,
        unsigned long *offset,
-       char **modname, char *namebuf);
+       char *modname, char *namebuf,
+       int safe_to_lock);

 /* Replace "%s" in format with address, if found */
 extern void __print_symbol(const char *fmt, unsigned long address);
@@ -42,7 +43,8 @@ static inline int kallsyms_lookup_size_o
 static inline const char *kallsyms_lookup(unsigned long addr,
        unsigned long *symbolsize,
        unsigned long *offset,
-       char **modname, char *namebuf)
+       char *modname, char *namebuf,
+       int safe_to_lock)
 {
  return NULL;
 }
diff --git a/include/linux/module.h b/include/linux/module.h
--- a/include/linux/module.h
+++ b/include/linux/module.h
@@ -370,10 +370,10 @@ struct module *module_text_address(unsig
 struct module *__module_text_address(unsigned long addr);
 int is_module_address(unsigned long addr);

-/* Returns module and fills in value, defined and namebuf, or NULL if
-   symnum out of range. */
-struct module *module_get_kallsym(unsigned int symnum, unsigned long *value,
-    char *type, char *name, size_t namelen);
+/* fills in value, type, name and module_name. returns 0 on success
+   or -ENOENT if symnum out of range. */
+int module_get_kallsym(unsigned int symnum, unsigned long *value,
+         char *type, char *name, char *module_name);

 /* Look for this name: can be of form module:name. */
 unsigned long module_kallsyms_lookup_name(const char *name);
@@ -451,11 +451,15 @@ do {                \
  }            \
 } while(0)

-/* For kallsyms to ask for address resolution.  NULL means not found. */
-const char *module_address_lookup(unsigned long addr,
-      unsigned long *symbolsize,
-      unsigned long *offset,
-      char **modname);
```

```
+/* For kallsyms to ask for address resolution. -ENOENT means not found.
+   if modname is NULL it is not filled with the module name
+   if safe_to_lock is 0, the function will not take the module_mutex
+   which can be useful during oops backtrace resolution */
+int module_address_lookup(unsigned long addr,
+     unsigned long *size,
+     unsigned long *offset,
+     char *symname, char *modname,
+     int safe_to_lock);

 /* For extable.c to search modules' exception tables. */
 const struct exception_table_entry *search_module_extables(unsigned long addr);
@@ -518,21 +522,21 @@ static inline void module_put(struct mod

 #define __unsafe(mod)

-/* For kallsyms to ask for address resolution.  NULL means not found. */
-static inline const char *module_address_lookup(unsigned long addr,
-      unsigned long *symbolsize,
-      unsigned long *offset,
-      char **modname)
+/* For kallsyms to ask for address resolution. -ENOENT means not found. */
+static inline int module_address_lookup(unsigned long addr,
+     unsigned long *size,
+     unsigned long *offset,
+     char *symname, char *modname
+     int safe_to_lock)
 {
- return NULL;
+ return -ENOENT;
 }

-static inline struct module *module_get_kallsym(unsigned int symnum,
-      unsigned long *value,
-      char *type, char *name,
-      size_t namelen)
+static inline int module_get_kallsym(unsigned int symnum,
+         unsigned long *value, char *type,
+         char *name, char *modname)
 {
- return NULL;
+ return -ENOENT;
 }

 static inline unsigned long module_kallsyms_lookup_name(const char *name)
diff --git a/kernel/kallsyms.c b/kernel/kallsyms.c
--- a/kernel/kallsyms.c
+++ b/kernel/kallsyms.c
```

```
@@ -229,7 +229,7 @@ int kallsyms_lookup_size_offset(unsigned
  if (is_ksym_addr(addr))
   return !!get_symbol_pos(addr, symbolsize, offset);

- return !!module_address_lookup(addr, symbolsize, offset, NULL);
+ return !module_address_lookup(addr, symbolsize, offset, NULL, NULL, 1);
 }

 /*
@@ -242,10 +242,9 @@ int kallsyms_lookup_size_offset(unsigned
 const char *kallsyms_lookup(unsigned long addr,
        unsigned long *symbolsize,
        unsigned long *offset,
-       char **modname, char *namebuf)
+       char *modname, char *namebuf,
+       int safe_to_lock)
 {
- const char *msym;
-
  namebuf[KSYM_NAME_LEN] = 0;
  namebuf[0] = 0;

@@ -255,36 +254,36 @@ const char *kallsyms_lookup(unsigned lon
   pos = get_symbol_pos(addr, symbolsize, offset);
   /* Grab name */
   kallsyms_expand_symbol(get_symbol_offset(pos), namebuf);
-  *modname = NULL;
+  if (modname)
+   *modname = '\0';
   return namebuf;
  }

  /* see if it's in a module */
- msym = module_address_lookup(addr, symbolsize, offset, modname);
- if (msym)
-  return strncpy(namebuf, msym, KSYM_NAME_LEN);
+ if (module_address_lookup(addr, symbolsize, offset, namebuf, modname, safe_to_lock) < 0)
+  return NULL;

- return NULL;
+ return namebuf;
 }

 /* Replace "%s" in format with address, or returns -errno. */
 void __print_symbol(const char *fmt, unsigned long address)
 {
- char *modname;
  const char *name;
```

```
  unsigned long offset, size;
  char namebuf[KSYM_NAME_LEN+1];
+ char modbuf[MODULE_NAME_LEN+1];
  char buffer[sizeof("%s+%#lx/%#lx [%s]") + KSYM_NAME_LEN +
        2*(BITS_PER_LONG*3/10) + MODULE_NAME_LEN + 1];

- name = kallsyms_lookup(address, &size, &offset, &modname, namebuf);
+ name = kallsyms_lookup(address, &size, &offset, modbuf, namebuf, 0);

  if (!name)
   sprintf(buffer, "0x%lx", address);
  else {
-  if (modname)
+  if (modbuf[0])
    sprintf(buffer, "%s+%#lx/%#lx [%s]", name, offset,
-    size, modname);
+    size, modbuf);
   else
    sprintf(buffer, "%s+%#lx/%#lx", name, offset, size);
  }
@@ -295,24 +294,24 @@ void __print_symbol(const char *fmt, uns
 struct kallsym_iter
 {
  loff_t pos;
- struct module *owner;
  unsigned long value;
  unsigned int nameoff; /* If iterating in core kernel symbols */
  char type;
  char name[KSYM_NAME_LEN+1];
+ char module_name[MODULE_NAME_LEN+1];
 };

 static int get_ksymbol_mod(struct kallsym_iter *iter)
 {
- iter->owner = module_get_kallsym(iter->pos - kallsyms_num_syms,
-      &iter->value, &iter->type,
-      iter->name, sizeof(iter->name));
- if (iter->owner == NULL)
-  return 0;
+ int err;

- /* Label it "global" if it is exported, "local" if not exported. */
- iter->type = is_exported(iter->name, iter->owner)
-  ? toupper(iter->type) : tolower(iter->type);
+ err = module_get_kallsym(iter->pos - kallsyms_num_syms,
+     &iter->value, &iter->type,
+     iter->name, iter->module_name);
+ if (err <= 0) {
```

```
+  iter->module_name[0] = '\0';
+  return 0;
+ }

  return 1;
 }
@@ -322,7 +321,7 @@ static unsigned long get_ksymbol_core(st
 {
  unsigned off = iter->nameoff;

- iter->owner = NULL;
+ iter->module_name[0] = '\0';
  iter->value = kallsyms_addresses[iter->pos];

  iter->type = kallsyms_get_symbol_type(off);
@@ -347,7 +346,7 @@ static int update_iter(struct kallsym_it
   iter->pos = pos;
   return get_ksymbol_mod(iter);
  }
-
+
  /* If we're not on the desired position, reset to new position. */
  if (pos != iter->pos)
   reset_iter(iter, pos);
@@ -382,15 +381,15 @@ static int s_show(struct seq_file *m, vo
 {
  struct kallsym_iter *iter = m->private;

- /* Some debugging symbols have no name.  Ignore them. */
+ /* Some debugging symbols have no name.  Ignore them. */
  if (!iter->name[0])
   return 0;

- if (iter->owner)
+ if (iter->module_name[0])
   seq_printf(m, "%0*lx %c %s\t[%s]\n",
       (int)(2*sizeof(void*)),
       iter->value, iter->type, iter->name,
-      module_name(iter->owner));
+      iter->module_name);
  else
   seq_printf(m, "%0*lx %c %s\n",
       (int)(2*sizeof(void*)),
diff --git a/kernel/kprobes.c b/kernel/kprobes.c
--- a/kernel/kprobes.c
+++ b/kernel/kprobes.c
@@ -837,7 +837,7 @@ static void __kprobes report_probe(struc
   kprobe_type = "k";
```

```
  if (sym)
   seq_printf(pi, "%p  %s  %s+0x%x  %s\n", p->addr, kprobe_type,
-   sym, offset, (modname ? modname : " "));
+   sym, offset, modname);
  else
   seq_printf(pi, "%p  %s  %p\n", p->addr, kprobe_type, p->addr);
 }
@@ -868,13 +868,13 @@ static int __kprobes show_kprobe_addr(st
  const char *sym = NULL;
  unsigned int i = *(loff_t *) v;
  unsigned long size, offset = 0;
- char *modname, namebuf[128];
+ char modname[MODULE_NAME_LEN+1], namebuf[KSYM_NAME_LEN+1];

  head = &kprobe_table[i];
  preempt_disable();
  hlist_for_each_entry_rcu(p, node, head, hlist) {
   sym = kallsyms_lookup((unsigned long)p->addr, &size,
-     &offset, &modname, namebuf);
+     &offset, modname, namebuf, 1);
   if (p->pre_handler == aggr_pre_handler) {
    list_for_each_entry_rcu(kp, &p->list, list)
     report_probe(pi, kp, sym, offset, modname);
diff --git a/kernel/lockdep.c b/kernel/lockdep.c
--- a/kernel/lockdep.c
+++ b/kernel/lockdep.c
@@ -342,9 +342,8 @@ static const char *usage_str[] =
 const char * __get_key_name(struct lockdep_subclass_key *key, char *str)
 {
  unsigned long offs, size;
- char *modname;

- return kallsyms_lookup((unsigned long)key, &size, &offs, &modname, str);
+ return kallsyms_lookup((unsigned long)key, &size, &offs, NULL, str, 0);
 }

 void
diff --git a/kernel/module.c b/kernel/module.c
--- a/kernel/module.c
+++ b/kernel/module.c
@@ -44,6 +44,8 @@
 #include <asm/semaphore.h>
 #include <asm/cacheflush.h>
 #include <linux/license.h>
+#include <linux/kallsyms.h>
+#include <linux/ctype.h>

 #if 0
```

```
 #define DEBUGP printk
@@ -2101,26 +2103,42 @@ static const char *get_ksymbol(struct mo
 /* For kallsyms to ask for address resolution.  NULL means not found.
    We don't lock, as this is used for oops resolution and races are a
    lesser concern. */
-const char *module_address_lookup(unsigned long addr,
+int module_address_lookup(unsigned long addr,
       unsigned long *size,
       unsigned long *offset,
-      char **modname)
+      char *symname, char *modname,
+      int safe_to_lock)
 {
  struct module *mod;
+ const char *modsym;
+ int ret = -ENOENT;

+ if (safe_to_lock)
+  mutex_lock(&module_mutex);
+
  list_for_each_entry(mod, &modules, list) {
   if (within(addr, mod->module_init, mod->init_size)
       || within(addr, mod->module_core, mod->core_size)) {
    if (modname)
-    *modname = mod->name;
-   return get_ksymbol(mod, addr, size, offset);
+    strlcpy(modname, mod->name, MODULE_NAME_LEN);
+   modsym = get_ksymbol(mod, addr, size, offset);
+   if (modsym) {
+    ret = 0;
+    if (symname)
+     strlcpy(symname, modsym, KSYM_NAME_LEN);
+   }
+   break;
   }
  }
- return NULL;
+
+ if (safe_to_lock)
+  mutex_unlock(&module_mutex);
+
+ return ret;
 }

-struct module *module_get_kallsym(unsigned int symnum, unsigned long *value,
-    char *type, char *name, size_t namelen)
+int module_get_kallsym(unsigned int symnum, unsigned long *value,
+   char *type, char *name, char *module_name)
```

```
 {
  struct module *mod;

@@ -2128,16 +2146,20 @@ struct module *module_get_kallsym(unsign
  list_for_each_entry(mod, &modules, list) {
   if (symnum < mod->num_symtab) {
    *value = mod->symtab[symnum].st_value;
-   *type = mod->symtab[symnum].st_info;
    strlcpy(name, mod->strtab + mod->symtab[symnum].st_name,
-    namelen);
+    KSYM_NAME_LEN);
+   strlcpy(module_name, mod->name, MODULE_NAME_LEN);
+   *type = mod->symtab[symnum].st_info;
+   /* Label it "global" if it is exported, "local" if not exported. */
+   *type = is_exported(name, mod) ? toupper(*type) : tolower(*type);
+
    mutex_unlock(&module_mutex);
-   return mod;
+   return 0;
   }
   symnum -= mod->num_symtab;
  }
  mutex_unlock(&module_mutex);
- return NULL;
+ return -ENOENT;
 }

 static unsigned long mod_find_symname(struct module *mod, const char *name)
diff --git a/kernel/time/timer_list.c b/kernel/time/timer_list.c
--- a/kernel/time/timer_list.c
+++ b/kernel/time/timer_list.c
@@ -42,9 +42,8 @@ static void print_name_offset(struct seq
  char namebuf[KSYM_NAME_LEN+1];
  unsigned long size, offset;
  const char *sym_name;
- char *modname;

- sym_name = kallsyms_lookup(addr, &size, &offset, &modname, namebuf);
+ sym_name = kallsyms_lookup(addr, &size, &offset, NULL, namebuf, 1);
  if (sym_name)
   SEQ_printf(m, "%s", sym_name);
  else
diff --git a/kernel/time/timer_stats.c b/kernel/time/timer_stats.c
--- a/kernel/time/timer_stats.c
+++ b/kernel/time/timer_stats.c
@@ -260,9 +260,8 @@ static void print_name_offset(struct seq
  char namebuf[KSYM_NAME_LEN+1];
  unsigned long size, offset;
```

```
  const char *sym_name;
- char *modname;

- sym_name = kallsyms_lookup(addr, &size, &offset, &modname, namebuf);
+ sym_name = kallsyms_lookup(addr, &size, &offset, NULL, namebuf, 1);
  if (sym_name)
   seq_printf(m, "%s", sym_name);
  else
diff --git a/mm/slab.c b/mm/slab.c
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -4380,16 +4380,16 @@ static void handle_slab(unsigned long *n
 static void show_symbol(struct seq_file *m, unsigned long address)
 {
 #ifdef CONFIG_KALLSYMS
- char *modname;
  const char *name;
  unsigned long offset, size;
  char namebuf[KSYM_NAME_LEN+1];
+ char modname[MODULE_NAME_LEN+1];

- name = kallsyms_lookup(address, &size, &offset, &modname, namebuf);
+ name = kallsyms_lookup(address, &size, &offset, modname, namebuf, 1);

  if (name) {
   seq_printf(m, "%s+%#lx/%#lx", name, offset, size);
-  if (modname)
+  if (modname[0])
    seq_printf(m, " [%s]", modname);
   return;
  }
```

Subject: Re: [PATCH RESEND 2/2] Fix some kallsyms_lookup() vs rmmod races
Posted by Ingo Molnar on Fri, 16 Mar 2007 16:18:53 GMT

View Forum Message <> Reply to Message

* Paulo Marques <pmarques@grupopie.com> wrote:

> >looking at the problem from another angle: wouldnt this be something
> >that would benefit from freeze_processes()/unfreeze_processes(), and
> >hence no locking would be required?
>
> I also considered this, but it seemed a little too "blunt" to stop
> everything (including completely unrelated processes and kernel
> threads) just to remove a module.

'just to remove a module' is very, very rare, on the timescale of most

kernel ops. Almost no distro does it. Furthermore, because we want to do
CPU-hotplug that way, we really want to make
freeze_processes()/unfreeze_processes() 'instantaneous' to the human -
and it is that already. (if it isnt in some case we can make it so)

Ingo

Subject: Re: [PATCH RESEND 2/2] Fix some kallsyms_lookup() vs rmmod races
Posted by Paulo Marques on Fri, 16 Mar 2007 17:16:39 GMT
View Forum Message <> Reply to Message

Ingo Molnar wrote:
> * Paulo Marques <pmarques@grupopie.com> wrote:
>
>>> looking at the problem from another angle: wouldnt this be something
>>> that would benefit from freeze_processes()/unfreeze_processes(), and
>>> hence no locking would be required?
>> I also considered this, but it seemed a little too "blunt" to stop
>> everything (including completely unrelated processes and kernel
>> threads) just to remove a module.
>
> 'just to remove a module' is very, very rare, on the timescale of most
> kernel ops. Almost no distro does it. Furthermore, because we want to do
> CPU-hotplug that way, we really want to make
> freeze_processes()/unfreeze_processes() 'instantaneous' to the human -
> and it is that already. (if it isnt in some case we can make it so)

Ok. I started to look at this approach and realized that module.c
already does this:

> ....
> static int __unlink_module(void *_mod)
> {
> struct module *mod = _mod;
> list_del(&mod->list);
> return 0;
> }
>
> /* Free a module, remove from lists, etc (must hold module mutex). */
> static void free_module(struct module *mod)
> {
> /* Delete from various lists */
> stop_machine_run(__unlink_module, mod, NR_CPUS);
> ....

However stop_machine_run doesn't seem like the right thing to do,
because users of the "modules" list don't seem to do anything to prevent

preemption. Am I missing something?

Does freeze_processes() / unfreeze_processes() solve this by only
freezing processes that have voluntarily scheduled (opposed to just
being preempted)?

--
Paulo Marques - www.grupopie.com

"The Computer made me do it."

Subject: Re: [PATCH RESEND 2/2] Fix some kallsyms_lookup() vs rmmod races
Posted by Andrew Morton on Fri, 16 Mar 2007 18:15:14 GMT
View Forum Message <> Reply to Message

On Fri, 16 Mar 2007 17:16:39 +0000 Paulo Marques <pmarques@grupopie.com> wrote:

> Does freeze_processes() / unfreeze_processes() solve this by only
> freezing processes that have voluntarily scheduled (opposed to just
> being preempted)?

It goes much much further than that. Those processes need to actually
perform an explicit call to try_to_freeze().

Subject: Re: [PATCH RESEND 2/2] Fix some kallsyms_lookup() vs rmmod races
Posted by Paulo Marques on Fri, 16 Mar 2007 20:27:29 GMT
View Forum Message <> Reply to Message

Andrew Morton wrote:
> On Fri, 16 Mar 2007 17:16:39 +0000 Paulo Marques <pmarques@grupopie.com> wrote:
>
>> Does freeze_processes() / unfreeze_processes() solve this by only
>> freezing processes that have voluntarily scheduled (opposed to just
>> being preempted)?
>
> It goes much much further than that. Those processes need to actually
> perform an explicit call to try_to_freeze().

Ok, I've just done a few tests with the attached patch. It basically
creates a freeze_machine_run function that is equivalent in interface to
stop_machine_run, but uses freeze_processes / thaw_processes to stop the
machine.

This is more of a proof of concept than an actual patch. At the very
least "freeze_machine_run" should be moved to kernel/power/process.c and

declared at include/linux/freezer.h so that it could be treated as a
more general purpose function and not something that is module specific.

Anyway, I then tested it by running a modprobe/rmmod loop while running
a "cat /proc/kallsyms" loop.

On the first run I forgot to remove the mutex_lock(module_mutex) from
the /proc/kallsyms read path and the freezer was unable to freeze the
"cat" process that was waiting for the same mutex that the freezer
process was holding :P

After removing the module_mutex locking from "module_get_kallsym"
everything was going fine (at least I got no oopses) until I got this:

kernel: Stopping user space processes timed out after 20 seconds (1
tasks refusing to freeze):
kernel: kbluetoothd
kernel: Restarting tasks ... <4> Strange, kseriod not stopped
kernel: Strange, pdflush not stopped
kernel: Strange, pdflush not stopped
kernel: Strange, kswapd0 not stopped
kernel: Strange, cifsoplockd not stopped
kernel: Strange, cifsdnotifyd not stopped
kernel: Strange, jfsIO not stopped
kernel: Strange, jfsCommit not stopped
kernel: Strange, jfsCommit not stopped
kernel: Strange, jfsSync not stopped
kernel: Strange, xfslogd/0 not stopped
kernel: Strange, xfslogd/1 not stopped
kernel: Strange, xfsdatad/0 not stopped
kernel: Strange, xfsdatad/1 not stopped
kernel: Strange, kjournald not stopped
kernel: Strange, khubd not stopped
kernel: Strange, khelper not stopped
kernel: Strange, kbluetoothd not stopped
kernel: done.

I repeated the test and did a Alt+SysRq+T to try to find out what
kbluetoothd was doing and got this:

kernel: kbluetoothd   D 79A11860   0 19156   1   19142
(NOTLB)
kernel: 9a269e4c 00000082 00000001 79a11860 00000000 79a09860 c7018030
00000003
kernel: 9a269e71 78475100 c7ebe000 c6730e40 00000000 00000001 00000001
00000001
kernel: 00000000 9a2d7570 79a11860 c7018140 00000000 00001832 42430d03
000000ab

kernel: Call Trace:
kernel: [<7845dba3>] wait_for_completion+0x7d/0xb7
kernel: [<781190ba>] default_wake_function+0x0/0xc
kernel: [<781190ba>] default_wake_function+0x0/0xc
kernel: [<7812c759>] call_usermodehelper_keys+0xd1/0xf1
kernel: [<7812c41e>] request_module+0x96/0xd9
kernel: [<783e30fe>] sock_alloc_inode+0x20/0x4e
kernel: [<78172559>] alloc_inode+0x15/0x115
kernel: [<78172d87>] new_inode+0x24/0x81
kernel: [<783e4003>] __sock_create+0x111/0x199
kernel: [<783e40a3>] sock_create+0x18/0x1d
kernel: [<783e40e1>] sys_socket+0x1c/0x43
kernel: [<783e51da>] sys_socketcall+0x247/0x24c
kernel: [<78121b2d>] sys_gettimeofday+0x2c/0x65
kernel: [<78103f10>] sysenter_past_esp+0x5d/0x81

And this was as far as I got...

This actually seems like a better approach than to hold module_mutex
everywhere to account for an operation that should be "rare" (module
loading/unloading). If something like this goes in, there are probably a
few more places inside module.c where we can drop the locking completely.

However, it still has a few gotchas. Apart from the problem above (which
may still be me doing something wrong) it makes module loading /
unloading depend on CONFIG_PM which is somewhat unexpected for the user.

Would it make sense to separate the process freezing / thawing API from
actual power management and create a new config option (CONFIG_FREEZER?)
that was automatically selected by the systems that used it (CONFIG_PM,
CONFIG_MODULES, etc.)? or is that overkill?

--
Paulo Marques - www.grupopie.com

"Nostalgia isn't what it used to be."

--- a/kernel/module.c
+++ b/kernel/module.c
@@ -35,7 +35,7 @@
 #include <linux/vermagic.h>
 #include <linux/notifier.h>
 #include <linux/sched.h>
-#include <linux/stop_machine.h>
+#include <linux/freezer.h>
 #include <linux/device.h>
 #include <linux/string.h>

```
 #include <linux/mutex.h>
@@ -618,13 +618,23 @@ static int __try_stop_module(void *_sref
  return 0;
 }

+static int freeze_machine_run(int (*fn)(void *), void *data, unsigned int cpu)
+{
+ int ret;
+ freeze_processes();
+ ret = fn(data);
+ thaw_processes();
+ return ret;
+}
+
 static int try_stop_module(struct module *mod, int flags, int *forced)
 {
  struct stopref sref = { mod, flags, forced };

- return stop_machine_run(__try_stop_module, &sref, NR_CPUS);
+ return freeze_machine_run(__try_stop_module, &sref, NR_CPUS);
 }

+
 unsigned int module_refcount(struct module *mod)
 {
  unsigned int i, total = 0;
@@ -1198,7 +1208,7 @@ static int __unlink_module(void *_mod)
 static void free_module(struct module *mod)
 {
  /* Delete from various lists */
- stop_machine_run(__unlink_module, mod, NR_CPUS);
+ freeze_machine_run(__unlink_module, mod, NR_CPUS);
  remove_sect_attrs(mod);
  mod_kobject_remove(mod);

@@ -1997,7 +2007,7 @@ sys_init_module(void __user *umod,

  /* Now sew it into the lists.  They won't access us, since
           strong_try_module_get() will fail. */
- stop_machine_run(__link_module, mod, NR_CPUS);
+ freeze_machine_run(__link_module, mod, NR_CPUS);

  /* Drop lock so they can recurse */
  mutex_unlock(&module_mutex);
@@ -2124,19 +2134,16 @@ struct module *module_get_kallsym(unsign
 {
  struct module *mod;
```

```
- mutex_lock(&module_mutex);
  list_for_each_entry(mod, &modules, list) {
   if (symnum < mod->num_symtab) {
    *value = mod->symtab[symnum].st_value;
    *type = mod->symtab[symnum].st_info;
    strlcpy(name, mod->strtab + mod->symtab[symnum].st_name,
     namelen);
-   mutex_unlock(&module_mutex);
    return mod;
   }
   symnum -= mod->num_symtab;
  }
- mutex_unlock(&module_mutex);
  return NULL;
 }
```

Subject: Re: [PATCH RESEND 2/2] Fix some kallsyms_lookup() vs rmmod races
Posted by Andrew Morton on Fri, 16 Mar 2007 20:49:49 GMT
View Forum Message <> Reply to Message

On Fri, 16 Mar 2007 20:27:29 +0000 Paulo Marques <pmarques@grupopie.com> wrote:

> Andrew Morton wrote:
> > On Fri, 16 Mar 2007 17:16:39 +0000 Paulo Marques <pmarques@grupopie.com> wrote:
> >
> >> Does freeze_processes() / unfreeze_processes() solve this by only
> >> freezing processes that have voluntarily scheduled (opposed to just
> >> being preempted)?
> >
> > It goes much much further than that. Those processes need to actually
> > perform an explicit call to try_to_freeze().
>
> Ok, I've just done a few tests with the attached patch. It basically
> creates a freeze_machine_run function that is equivalent in interface to
> stop_machine_run, but uses freeze_processes / thaw_processes to stop the
> machine.
>
> This is more of a proof of concept than an actual patch. At the very
> least "freeze_machine_run" should be moved to kernel/power/process.c and
> declared at include/linux/freezer.h so that it could be treated as a
> more general purpose function and not something that is module specific.

OK.

> Anyway, I then tested it by running a modprobe/rmmod loop while running
> a "cat /proc/kallsyms" loop.
>

> On the first run I forgot to remove the mutex_lock(module_mutex) from
> the /proc/kallsyms read path and the freezer was unable to freeze the
> "cat" process that was waiting for the same mutex that the freezer
> process was holding :P
>
> After removing the module_mutex locking from "module_get_kallsym"
> everything was going fine (at least I got no oopses) until I got this:
>
> kernel: Stopping user space processes timed out after 20 seconds (1
> tasks refusing to freeze):
> kernel: kbluetoothd
> kernel: Restarting tasks ... <4> Strange, kseriod not stopped
> kernel: Strange, pdflush not stopped
> kernel: Strange, pdflush not stopped
> kernel: Strange, kswapd0 not stopped
> kernel: Strange, cifsoplockd not stopped
> kernel: Strange, cifsdnotifyd not stopped
> kernel: Strange, jfsIO not stopped
> kernel: Strange, jfsCommit not stopped
> kernel: Strange, jfsCommit not stopped
> kernel: Strange, jfsSync not stopped
> kernel: Strange, xfslogd/0 not stopped
> kernel: Strange, xfslogd/1 not stopped
> kernel: Strange, xfsdatad/0 not stopped
> kernel: Strange, xfsdatad/1 not stopped
> kernel: Strange, kjournald not stopped
> kernel: Strange, khubd not stopped
> kernel: Strange, khelper not stopped
> kernel: Strange, kbluetoothd not stopped
> kernel: done.

There are a bunch of freezer fixes in -mm. But problems might still remain
- I don't think freezer has had a lot of load put on it yet, but it will
soon and it needs to become reliable.

> I repeated the test and did a Alt+SysRq+T to try to find out what
> kbluetoothd was doing and got this:
>
> kernel: kbluetoothd   D 79A11860   0 19156   1   19142
> (NOTLB)
> kernel: 9a269e4c 00000082 00000001 79a11860 00000000 79a09860 c7018030
> 00000003
> kernel: 9a269e71 78475100 c7ebe000 c6730e40 00000000 00000001 00000001
> 00000001
> kernel: 00000000 9a2d7570 79a11860 c7018140 00000000 00001832 42430d03
> 000000ab
> kernel: Call Trace:

> kernel: [<7845dba3>] wait_for_completion+0x7d/0xb7
> kernel: [<781190ba>] default_wake_function+0x0/0xc
> kernel: [<781190ba>] default_wake_function+0x0/0xc
> kernel: [<7812c759>] call_usermodehelper_keys+0xd1/0xf1
> kernel: [<7812c41e>] request_module+0x96/0xd9
> kernel: [<783e30fe>] sock_alloc_inode+0x20/0x4e
> kernel: [<78172559>] alloc_inode+0x15/0x115
> kernel: [<78172d87>] new_inode+0x24/0x81
> kernel: [<783e4003>] __sock_create+0x111/0x199
> kernel: [<783e40a3>] sock_create+0x18/0x1d
> kernel: [<783e40e1>] sys_socket+0x1c/0x43
> kernel: [<783e51da>] sys_socketcall+0x247/0x24c
> kernel: [<78121b2d>] sys_gettimeofday+0x2c/0x65
> kernel: [<78103f10>] sysenter_past_esp+0x5d/0x81
>
> And this was as far as I got...
>
> This actually seems like a better approach than to hold module_mutex
> everywhere to account for an operation that should be "rare" (module
> loading/unloading). If something like this goes in, there are probably a
> few more places inside module.c where we can drop the locking completely.

Yes, using the freezer and module load/unload time seems like a good idea.

> However, it still has a few gotchas. Apart from the problem above (which
> may still be me doing something wrong) it makes module loading /
> unloading depend on CONFIG_PM which is somewhat unexpected for the user.

yup.

> Would it make sense to separate the process freezing / thawing API from
> actual power management and create a new config option (CONFIG_FREEZER?)
> that was automatically selected by the systems that used it (CONFIG_PM,
> CONFIG_MODULES, etc.)? or is that overkill?

Yes, freezer needs to be decoupled from swsusp and from power management
and it should become a first-class core kernel component. Whether we would
need a CONFIG_FREEZER isn't clear - I suspect we'd end up just compiling it
unconditionally.

I cc'ed Rafael, who is doing the freezer revamp work.

Subject: Re: [PATCH RESEND 2/2] Fix some kallsyms_lookup() vs rmmod races
Posted by Rusty Russell on Sat, 17 Mar 2007 09:32:51 GMT
View Forum Message <> Reply to Message

On Fri, 2007-03-16 at 14:44 +0300, Alexey Dobriyan wrote:

> [cc'ing folks whose proc files are affected]
>
> kallsyms_lookup() can call module_address_lookup() which iterates over
> modules list without module_mutex taken. Comment at the top of
> module_address_lookup() says it's for oops resolution so races are
> irrelevant, but in some cases it's reachable from regular code:

Yes, this changed somewhere along the way.

I prefer keeping the lock internal as much as possible, and have the
crash code use an __ variant of the function.

Note also that it might be an idea to have less-powerful accessors than
kallsyms_lookup...

Thanks!
Rusty.

Subject: Re: [PATCH RESEND 2/2] Fix some kallsyms_lookup() vs rmmod races
Posted by Rusty Russell on Sat, 17 Mar 2007 09:37:18 GMT
View Forum Message <> Reply to Message

On Fri, 2007-03-16 at 12:51 +0100, Ingo Molnar wrote:
> * Alexey Dobriyan <adobriyan@sw.ru> wrote:
>
> > [cc'ing folks whose proc files are affected]
> >
> > kallsyms_lookup() can call module_address_lookup() which iterates over
> > modules list without module_mutex taken. Comment at the top of
> > module_address_lookup() says it's for oops resolution so races are
> > irrelevant, but in some cases it's reachable from regular code:
>
> looking at the problem from another angle: wouldnt this be something
> that would benefit from freeze_processes()/unfreeze_processes(), and
> hence no locking would be required?

Actually, the list manipulation is done with stop_machine for this
reason. Alexey, is preempt enabled in your kernel?

Rusty.

Subject: Re: [PATCH RESEND 2/2] Fix some kallsyms_lookup() vs rmmod races
Posted by Rusty Russell on Sat, 17 Mar 2007 10:36:33 GMT
View Forum Message <> Reply to Message

On Fri, 2007-03-16 at 12:49 -0800, Andrew Morton wrote:
> > Ok, I've just done a few tests with the attached patch. It basically
> > creates a freeze_machine_run function that is equivalent in interface to
> > stop_machine_run, but uses freeze_processes / thaw_processes to stop the
> > machine.

I've never been convinced that the freezer was a good idea.
stop_machine is a damn big hammer, but it works.

Rusty.

Subject: Re: [PATCH RESEND 2/2] Fix some kallsyms_lookup() vs rmmod races
Posted by Alexey Dobriyan on Mon, 19 Mar 2007 09:50:26 GMT
View Forum Message <> Reply to Message

On Fri, Mar 16, 2007 at 08:27:29PM +0000, Paulo Marques wrote:
> Andrew Morton wrote:
> >On Fri, 16 Mar 2007 17:16:39 +0000 Paulo Marques <pmarques@grupopie.com>
> >wrote:
> >
> >>Does freeze_processes() / unfreeze_processes() solve this by only
> >>freezing processes that have voluntarily scheduled (opposed to just
> >>being preempted)?
> >
> >It goes much much further than that. Those processes need to actually
> >perform an explicit call to try_to_freeze().
>
> Ok, I've just done a few tests with the attached patch. It basically
> creates a freeze_machine_run function that is equivalent in interface to
> stop_machine_run, but uses freeze_processes / thaw_processes to stop the
> machine.
>
> This is more of a proof of concept than an actual patch. At the very
> least "freeze_machine_run" should be moved to kernel/power/process.c and
> declared at include/linux/freezer.h so that it could be treated as a
> more general purpose function and not something that is module specific.
>
> Anyway, I then tested it by running a modprobe/rmmod loop while running
> a "cat /proc/kallsyms" loop.
>
> On the first run I forgot to remove the mutex_lock(module_mutex) from
> the /proc/kallsyms read path and the freezer was unable to freeze the
> "cat" process that was waiting for the same mutex that the freezer
> process was holding :P
>
> After removing the module_mutex locking from "module_get_kallsym"
> everything was going fine (at least I got no oopses) until I got this:

>
> kernel: Stopping user space processes timed out after 20 seconds (1
> tasks refusing to freeze):
> kernel: kbluetoothd
> kernel: Restarting tasks ... <4> Strange, kseriod not stopped
> kernel: Strange, pdflush not stopped
> kernel: Strange, pdflush not stopped
> kernel: Strange, kswapd0 not stopped
> kernel: Strange, cifsoplockd not stopped
> kernel: Strange, cifsdnotifyd not stopped
> kernel: Strange, jfsIO not stopped
> kernel: Strange, jfsCommit not stopped
> kernel: Strange, jfsCommit not stopped
> kernel: Strange, jfsSync not stopped
> kernel: Strange, xfslogd/0 not stopped
> kernel: Strange, xfslogd/1 not stopped
> kernel: Strange, xfsdatad/0 not stopped
> kernel: Strange, xfsdatad/1 not stopped
> kernel: Strange, kjournald not stopped
> kernel: Strange, khubd not stopped
> kernel: Strange, khelper not stopped
> kernel: Strange, kbluetoothd not stopped
> kernel: done.
>
> I repeated the test and did a Alt+SysRq+T to try to find out what
> kbluetoothd was doing and got this:
>
> kernel: kbluetoothd   D 79A11860   0 19156   1   19142
> (NOTLB)
> kernel: 9a269e4c 00000082 00000001 79a11860 00000000 79a09860 c7018030
> 00000003
> kernel: 9a269e71 78475100 c7ebe000 c6730e40 00000000 00000001 00000001
> 00000001
> kernel: 00000000 9a2d7570 79a11860 c7018140 00000000 00001832 42430d03
> 000000ab
> kernel: Call Trace:
> kernel: [<7845dba3>] wait_for_completion+0x7d/0xb7
> kernel: [<781190ba>] default_wake_function+0x0/0xc
> kernel: [<781190ba>] default_wake_function+0x0/0xc
> kernel: [<7812c759>] call_usermodehelper_keys+0xd1/0xf1
> kernel: [<7812c41e>] request_module+0x96/0xd9
> kernel: [<783e30fe>] sock_alloc_inode+0x20/0x4e
> kernel: [<78172559>] alloc_inode+0x15/0x115
> kernel: [<78172d87>] new_inode+0x24/0x81
> kernel: [<783e4003>] __sock_create+0x111/0x199
> kernel: [<783e40a3>] sock_create+0x18/0x1d
> kernel: [<783e40e1>] sys_socket+0x1c/0x43
> kernel: [<783e51da>] sys_socketcall+0x247/0x24c

> kernel: [<78121b2d>] sys_gettimeofday+0x2c/0x65
> kernel: [<78103f10>] sysenter_past_esp+0x5d/0x81
>
> And this was as far as I got...
>
> This actually seems like a better approach than to hold module_mutex
> everywhere to account for an operation that should be "rare" (module
> loading/unloading). If something like this goes in, there are probably a
> few more places inside module.c where we can drop the locking completely.
>
> However, it still has a few gotchas. Apart from the problem above (which
> may still be me doing something wrong) it makes module loading /
> unloading depend on CONFIG_PM which is somewhat unexpected for the user.

It'd be a bug. cat /proc/kallsyms should work reliably regardless of
CONFIG_PM, CONFIG_MODULES, etc.

> Would it make sense to separate the process freezing / thawing API from
> actual power management and create a new config option (CONFIG_FREEZER?)
> that was automatically selected by the systems that used it (CONFIG_PM,
> CONFIG_MODULES, etc.)? or is that overkill?

I tried you patch on top of 2.6.21-rc4-5851fadce8824d5d4b8fd02c22ae098401f6489e
shrug
Let's say that is doesn't work here. :)

On boot I got
```
-------------------------------
Stopping user space processes timed out after 20 seconds (1 tasks refusing to freeze):
 mount
 Strange, kseriod not stopped
 Strange, pdflush not stopped
 Strange, pdflush not stopped
 Strange, kswapd0 not stopped
 Strange, kjournald not stopped
 Strange, khelper not stopped
 Strange, mount not stopped
Filesystem "loop0": Disabling barriers, not supported by the underlying device
XFS mounting filesystem loop0
Ending clean XFS mount for filesystem: loop0
-----------------------------------------------
Stopping user space processes timed out after 20 seconds (1 tasks refusing to freeze):
 dhcpcd
 Strange, kseriod not stopped
 Strange, pdflush not stopped
 Strange, pdflush not stopped
 Strange, kswapd0 not stopped
 Strange, kjournald not stopped
```

Strange, xfslogd/0 not stopped
Strange, xfsdatad/0 not stopped
Strange, xfsbufd not stopped
Strange, xfssyncd not stopped
Strange, khubd not stopped
Strange, khelper not stopped
Strange, dhcpcd not stopped
NET: Registered protocol family 17
ohci_hcd: 2006 August 04 USB 1.1 'Open' Host Controller (OHCI) Driver
==========

Now it booted (slowly) but running fine. Time to test cat /proc/kallsyms.
rmmod hangs in D-state (probably immediately)

==========
rmmod D 00000046 2724 6868 6861 (NOTLB)
f3640f14 00000086 c03e213c 00000046 00000000 00000000 c0465408 00000005
c1b74070 9091506d 0000001c 0000e340 c1b74190 c04653e8 00000046 c04653ec
c04653e8 c04653ec c04653e8 f3640f28 f3640f3c c02d260c 00000001 c1b74070
Call Trace:
[<c02d260c>] wait_for_completion+0x9e/0xb7
[<c0113f67>] default_wake_function+0x0/0xc
[<c0127f72>] kthread_stop+0x4c/0x6a
[<c0125470>] destroy_workqueue+0x24/0x54
[<f94deada>] xfs_buf_terminate+0x14/0x2d [xfs]
[<f94e6dc9>] exit_xfs_fs+0x19/0x27 [xfs]
[<c0138662>] sys_delete_module+0x11e/0x17e
[<c012b051>] up_read+0x14/0x27
[<c0111ea4>] do_page_fault+0x311/0x5ad
[<c0103e72>] sysenter_past_esp+0x5f/0x99
==========
Also following two lines were attached at the end of the Alt+SysRq+T output

Clocksource tsc unstable (delta = 23420074248 ns)
Time: acpi_pm clocksource has been installed.

Subject: Re: [PATCH RESEND 2/2] Fix some kallsyms_lookup() vs rmmod races
Posted by Alexey Dobriyan on Mon, 19 Mar 2007 10:14:25 GMT
View Forum Message <> Reply to Message

On Sat, Mar 17, 2007 at 08:37:18PM +1100, Rusty Russell wrote:
> On Fri, 2007-03-16 at 12:51 +0100, Ingo Molnar wrote:
> > * Alexey Dobriyan <adobriyan@sw.ru> wrote:
> >
> > > [cc'ing folks whose proc files are affected]
> > >
> > > kallsyms_lookup() can call module_address_lookup() which iterates over

> > > modules list without module_mutex taken. Comment at the top of
> > > module_address_lookup() says it's for oops resolution so races are
> > > irrelevant, but in some cases it's reachable from regular code:
> >
> > looking at the problem from another angle: wouldnt this be something
> > that would benefit from freeze_processes()/unfreeze_processes(), and
> > hence no locking would be required?
>
> Actually, the list manipulation is done with stop_machine for this
> reason.

mmm, my changelog is slightly narrow than it should be.

Non-emergency code is traversing modules list.
It finds "struct module *".
module is removed.
"struct module *" is now meaningless, but still dereferenced.

How would all this refrigerator stuff would help? It wouldn't,

Non-emergency code is traversing modules list.
It finds "struct module *".
Everything is freezed.
Module is removed.
Everything is unfreezed.
"struct module *" is now meaningless, but still dereferenced.

> Alexey, is preempt enabled in your kernel?

Yes. FWIW,

CONFIG_PREEMPT=y
CONFIG_PREEMPT_BKL=y
CONFIG_DEBUG_PREEMPT=y

I very much agree with proto-patch which _copies_ all relevant
information into caller-supplied structure, keeping module_mutex private.
Time to split it sanely.

Subject: Re: [PATCH RESEND 2/2] Fix some kallsyms_lookup() vs rmmod races
Posted by Paulo Marques on Mon, 19 Mar 2007 15:17:17 GMT
View Forum Message <> Reply to Message

Alexey Dobriyan wrote:
> On Sat, Mar 17, 2007 at 08:37:18PM +1100, Rusty Russell wrote:
>> On Fri, 2007-03-16 at 12:51 +0100, Ingo Molnar wrote:
>>> [...]

>>> looking at the problem from another angle: wouldnt this be something
>>> that would benefit from freeze_processes()/unfreeze_processes(), and
>>> hence no locking would be required?
>> Actually, the list manipulation is done with stop_machine for this
>> reason.
>
> mmm, my changelog is slightly narrow than it should be.
>
> Non-emergency code is traversing modules list.
> It finds "struct module *".
> module is removed.
> "struct module *" is now meaningless, but still dereferenced.
>
> How would all this refrigerator stuff would help? It wouldn't,
>
> Non-emergency code is traversing modules list.
> It finds "struct module *".
> Everything is freezed.
> Module is removed.
> Everything is unfreezed.
> "struct module *" is now meaningless, but still dereferenced.

That is why I asked if the refrigerator would preempt processes or not.
AFAICS there is no path where the "struct module *" that is returned is
used after a voluntary preemption point, so it should be safe. I might
be missing something, though.

However, this isn't very robust. Since the functions are still returning
pointers to module data, some changes in the future might keep the
pointer, and use it after a valid freezing point -> oops.

>> Alexey, is preempt enabled in your kernel?
>
> Yes. FWIW,
>
> CONFIG_PREEMPT=y
> CONFIG_PREEMPT_BKL=y
> CONFIG_DEBUG_PREEMPT=y
>
> I very much agree with proto-patch which _copies_ all relevant
> information into caller-supplied structure, keeping module_mutex private.
> Time to split it sanely.

That depends on the roadmap: if we think the freezer approach is the
best in the long run, maybe your less intrusive (in the sense that it
changes less stuff) patch should go in now (as a quick fix to mainline)
so that after we've sorted out the bugs from the freezer in -mm, it will
be easier to revert.

However, if we think the most reliable solution would be to not return
internal module information through pointers and keep all that logic
internal to module.c, then the "proto-patch" with some improvements
might be the way to go...

--
Paulo Marques - www.grupopie.com

"God is love. Love is blind. Ray Charles is blind. Ray Charles is God."

Subject: Re: [PATCH RESEND 2/2] Fix some kallsyms_lookup() vs rmmod races
Posted by Rusty Russell on Mon, 19 Mar 2007 23:23:07 GMT
View Forum Message <> Reply to Message

On Mon, 2007-03-19 at 13:21 +0300, Alexey Dobriyan wrote:
> I very much agree with proto-patch which _copies_ all relevant
> information into caller-supplied structure, keeping module_mutex private.
> Time to split it sanely.

Indeed. The current interface needs to be ripped apart and put together
sanely.

Thanks!
Rusty.
