Modules in 2.6: Breaking The Kernel, and What I Learned

Rusty Russell

IBM Linux Technology Center, OzLabs
Canberra, Australia

rusty@rustcorp.com.au
rusty@au.ibm.com

Contents

- Time Better Spent
- Who is Rusty Russell?
- 2001: A Small Problem
- 31 October 2002: Kernel Freeze
- Richard Henderson
- David S. Miller
- New Features
- Conclusion

Time Better Spent

- Read The Lions Book
- Read the original TDB source code.
- Read “Elements of Programming Style”
- Become familiar with the following software:
  - gperf
  - qemu
  - rsync
  - valgrind
  - ccache
  - distcc

Who Is Rusty Russell?

- Linux Kernel Programmer
  - ipchains, netfilter, futexes, per-cpu variables, modules,...
- Author of the original Linux Graphing Project
  - http://fcgp.sourceforge.net/
- Organizer of the first Australian Linux Conference
  - http://www.linux.org.au/projects/calu/
- Author of Networking Concepts HOWTO
  - http://www.netfilter.org/unreliable-guides/
- KernelTrap Interview:
  - http://kerneltrap.org/node/view/892

2001: A Small Project

January 2001: Connection Tracking

- ip_conntrack: a module for tracking IP network connections.
- All modules have a usage count.
- ip_conntrack usage count always zero
  - When removed, waited (sometimes forever!) until all connections ended.
- Making reference count increase per connection
  - Slow
  - Would make the module unremovable for most people.
- A new module unload method was needed.

March 2001: The Module Code

- How hard would it be to modify the module code?

Rusty's Lesson #1

Many Major Projects Start With:
“I Only Need To Change This One Thing...”

The Old Module Code

- Inserting a module is done as follows:
  - query_module(QM_MODULES) to return list of modules
  - query_module(QM_INFO) on each one to see if it's active
  - query_module(QM_SYMBOLS) to get the values of symbols exported by that module.
  - create_module(name, size) to get the address of the module.
  - Do module linking for that architecture.
    - Perform relocations
    - List dependencies in header
  - Call init_module(name, struct module)
    - Kernel verifies structure.
    - Kernel attaches dependencies

The Old Module Code

- Changing the code is difficult, since userspace (modutils) and kernel distributed separately.
- For example, to add a field to the module structure
  - Inside the kernel, use mod_member_present() to detect old userspace, and deal with it.
  - Inside userspace, detect old kernel versions and deal with them.

July 2001: Some New Module Code

- I experimented with doing the linking inside the kernel.
- Inserting a module is done as follows:
  - Call init_module(pointer, size, option-string)
    - Kernel resolves symbols and dependencies
    - Kernel parses user options
    - Kernel calls init function.
- Old insmod: 7103 lines of code

New Insmod

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <fcntl.h>

int main(int argc, char *argv[])
{
        int fd = 0;
        void *p;
        struct stat st;

        if (argc > 1)
                fd = open(argv[1], O_RDONLY);

        fstat(fd, &st);
        p = mmap(NULL, st.st_size, PROT_READ, MAP_PRIVATE, fd, 0);
        return syscall(__NR_init_module, p, st.st_size, argc>2 ? argv[2] : "");
}
```

Kernel Size

- Userspace gets smaller, but how much larger is the kernel?
- 2.4 module code: 1284 lines
  - create_module - 50 lines
  - init_module - 230 lines
  - delete_module - 80 lines
  - query_module - 270 lines
  - /proc/modules - 80 lines
  - /proc/ksyms - 80 lines

Diagram of Old Module Code

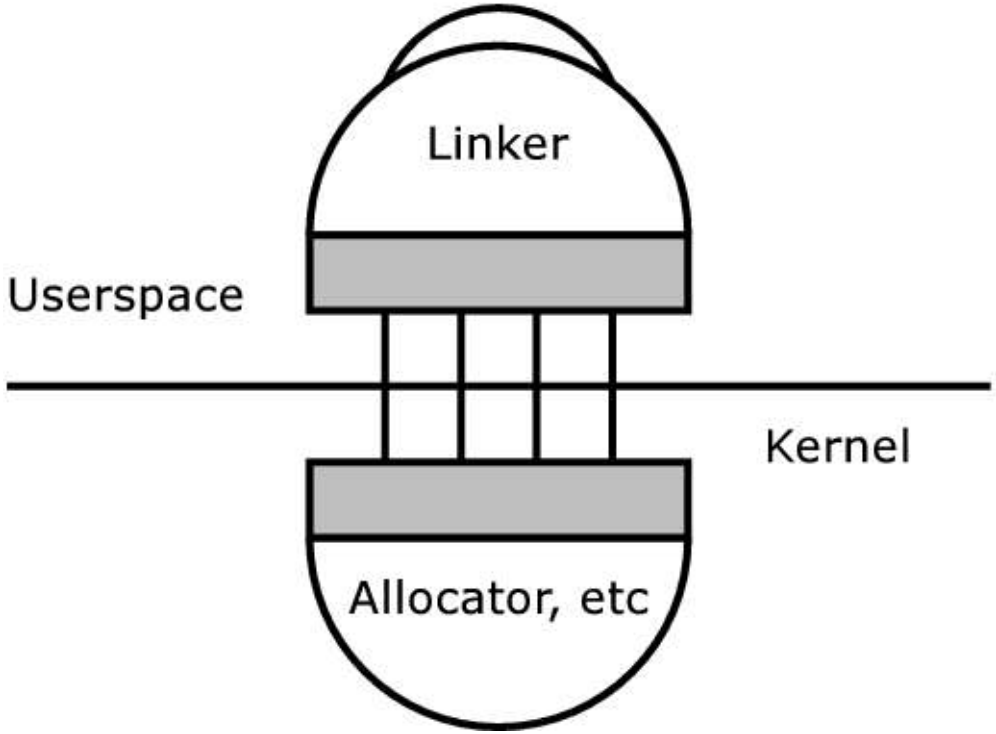
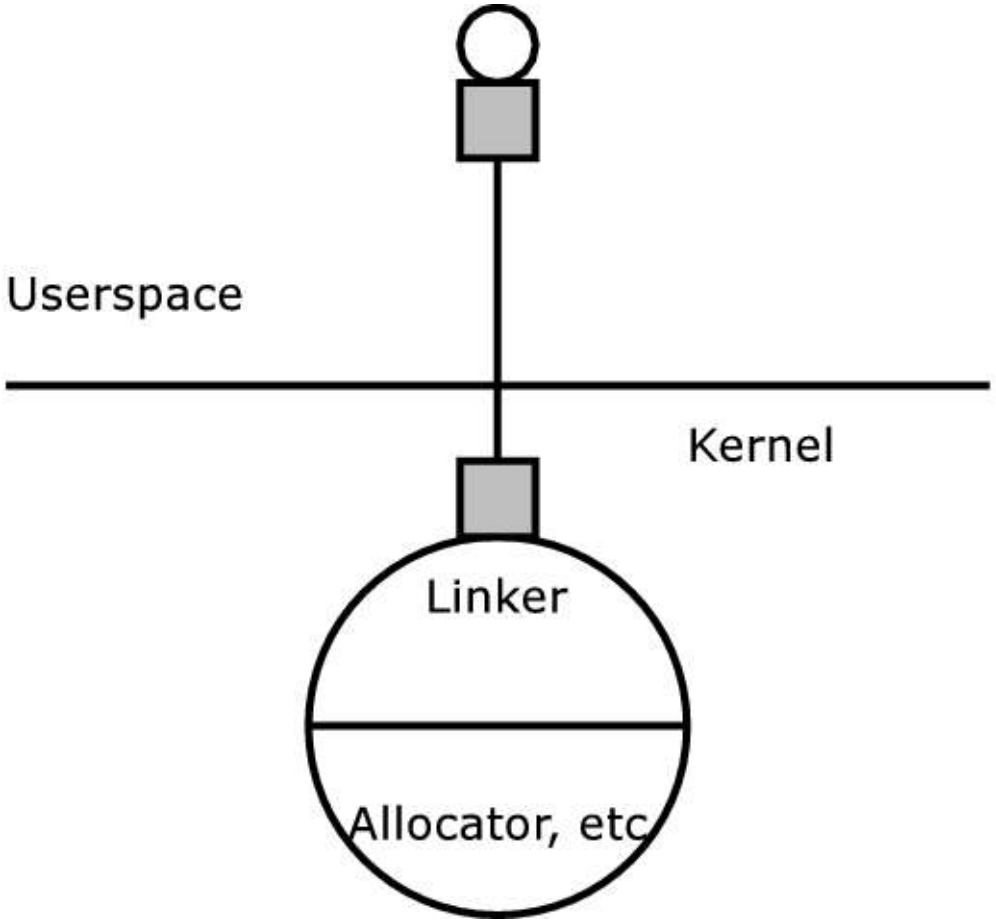


Diagram of New Module Code



Kernel Size

- 2.4 module code: 1284 lines
- 2.6 module code: 1157 lines + arch code
- 2.6 i386 arch code: 130 lines (smallest)
- 2.6 ia64 arch code: 875 lines (largest)

Rusty's Lesson #2

You'll Only Ever Know If You Write The Code.

October 31 2002: Kernel Freeze

October 2002: Kernel Freeze

- As organized at the kernel summit, the kernel entered Feature Freeze on October 31, 2002.
  - The new module code was not in the kernel.

November 2002: Module Merge

- At breakfast before leaving Japan (for Spain), Anton Blanchard tells me Linus has included my patch.
  - But module parameter code is not included
  - Userspace utilities are primitive (no modprobe)
  - Only i386 works at all
- I didn't get much sleep in Spain.

Rusty's Lesson #3

Linus Always Chooses The Worst Time To Apply Your Patch.

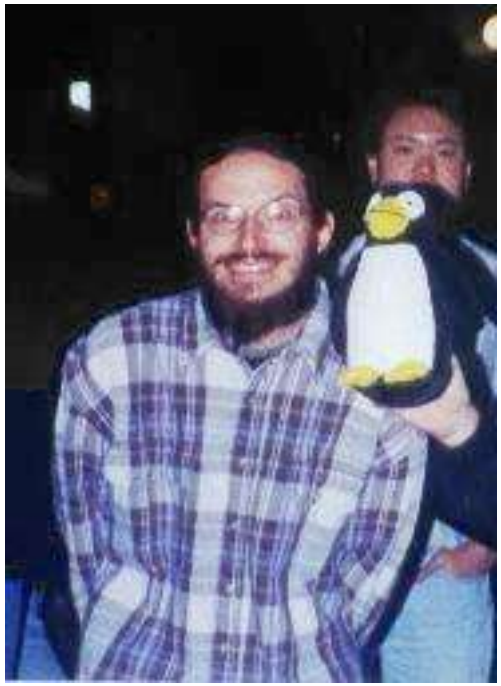
The Great Module War

It's still in flux, as Rusty combines world-wide travel plus frantic bug-fixing as he is being pursued (virtually) around the world by hordes of angry kernel developers.

-- Theodore Ts'o, Linux Kernel Mailing List

November 2002: David S Miller

- Dave Miller is the Linux Networking Maintainer.
- Dave Miller is the Sparc64 Architecture Maintainer.
- Dave Miller did the original SPARCLinux port.

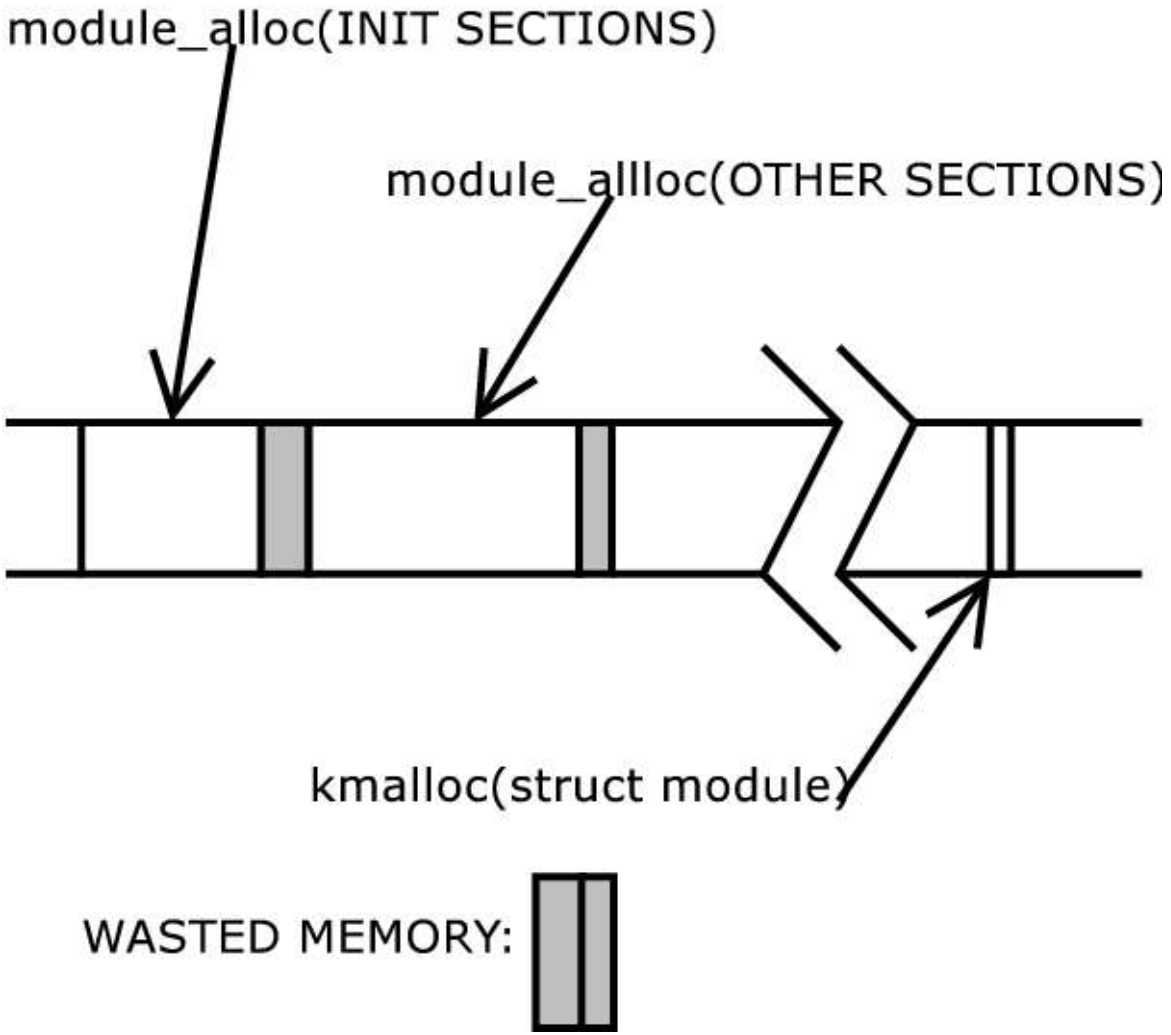


David S. Miller

- Sparc and Sparc64 need modules placed in the lowest 2GB of memory.
  - I knew this
  - Architectures defined “module_alloc” which the module code calls to allocate space for the two parts of the module
    - The init code (to be discarded after initialization)
    - The rest of the module
  - The module structure itself was called using the normal “kmalloc” inside the kernel.

David S Miller

- The Sparc allocator always allocates pages (4096 bytes):



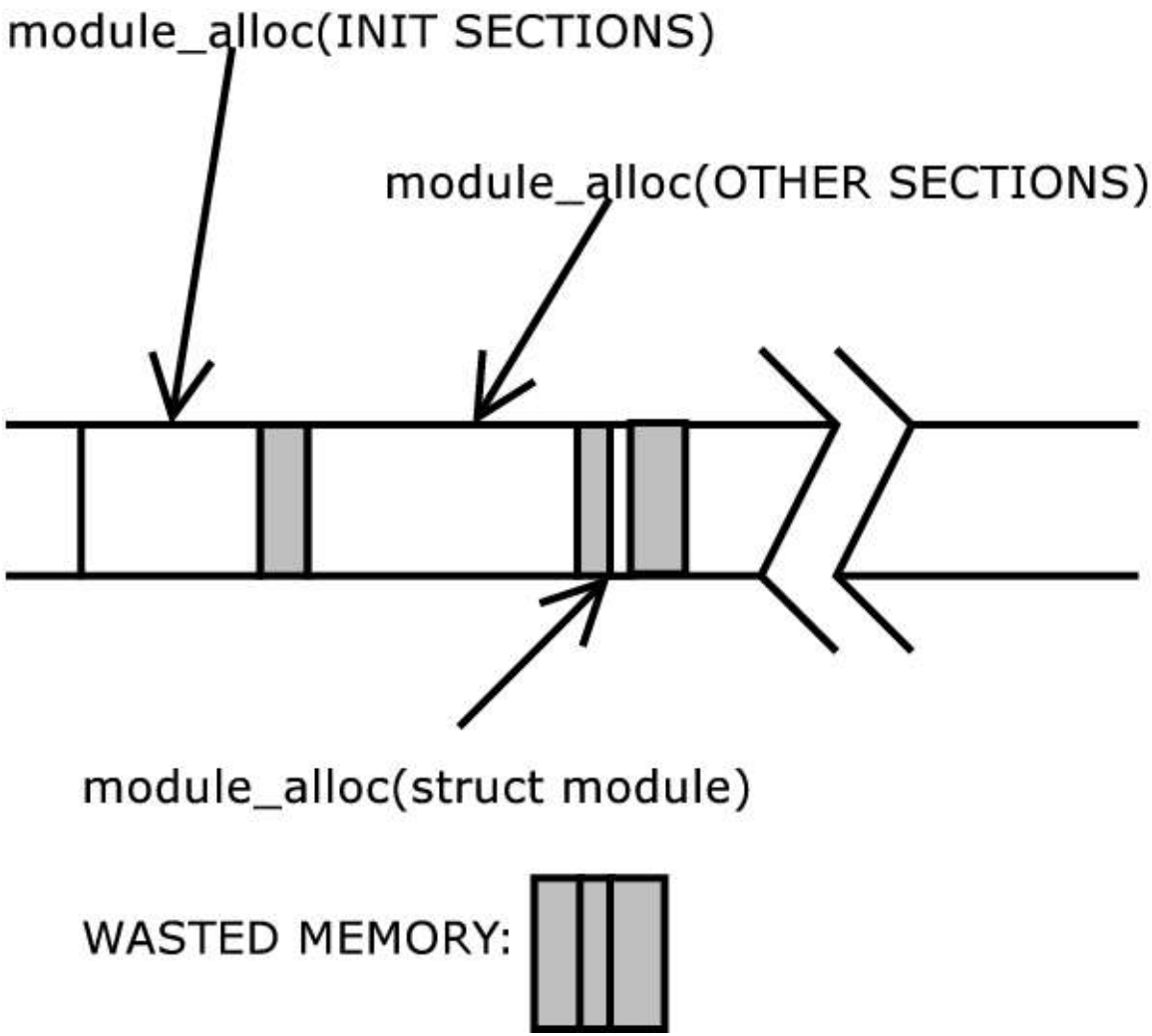
David S Miller

To: rusty@rustcorp.com.au
Subject: Re: new module stuff
From: "David S. Miller" <davem@redhat.com>

Dude, you have to allocate the struct module in the module section just like the old code. I build sparc64 kernel modules into the lower 32-bits of the kernel address space, and if __init_module is kmalloc()'d it can't be relocated properly.

David S Miller

- The obvious solution:



David S Miller

From: Rusty Russell <rusty@rustcorp.com.au>
To: "David S. Miller" <davem@redhat.com>
Subject: Re: new module stuff

In message <20021114.100125.118043272.davem@redhat.com> you write:
> How about this, in the module_alloc call you allocate size + sizeof(*mod)

I already tack the user-supplied options on the end of the module, so your optimization interferes with my optimization 8)

I *will* code you a solution: struct module on sparc64 is 2368 bytes for NR_CPUS=32 (384 for UP).

You deserve it for just getting down and helping code, rather than bitching about breakage 8)

Thanks!
Rusty.

David S Miller

To: rusty@rustcorp.com.au
Subject: Re: new module stuff
From: "David S. Miller" <davem@redhat.com>

I'm not going to argue about 1 page for now, implement this fix however you want and then we'll revisit this later. :-)

Rusty's Lesson #4

Dave Miller is Cool.

Rusty's Lesson #4a

Architecture Maintainers Are Some of The Smartest (and Nicest) Programmers To Work With.

November 2002: Richard Henderson

- Richard Henderson is the Alpha Linux Maintainer.
- Richard Henderson is a GCC maintainer.
- Richard Henderson gave the keynote at the GCC summit.

December 2002: Richard Henderson

- Richard Henderson reported some problems he found in my in-kernel module loader.
  - He wrote some beautiful patches which I took.

One more thing:

Are you really REALLY sure you don't want to load ET_DYN or ET_EXEC files (aka shared libraries or executables) instead of ET_REL files (aka .o files)?

December 2002: Richard Henderson

- It turns out that shared objects are much simpler to load than .o files
  - Normally .so (ET_DYN) objects are compiled to be position independent (-fPIC).
    - This makes them slightly slower than normal code.
  - But they don't have to be.

Rusty's Lesson #5

Ideas From Old Code Stay In Your Brain.

December 2002: Richard Henderson

From: Rusty Russell <rusty@rustcorp.com.au>
To: Richard Henderson <rth@twiddle.net>
Cc: linux-kernel@vger.kernel.org
Subject: Re: in-kernel linking issues

In message <20021115142226.B25624@twiddle.net> you write:
> You've only got two relocation types, you don't need to worry about
> .got, .plt, .opd allocation, nor sorting sections into a required
> order, nor sorting COMMON symbols.

Hmm, OK, I guess this is where I say "patch welcome"?

Rusty.

Rusty's Lesson #6

Always Ask For A Patch
(David Miller Taught Me This)

December 2002: Richard Henderson

- We spent about a month working on using shared libraries, including porting to all the architectures.
  - Turned out to need a minor toolchain change on amd64, and a major change on MIPS.
  - No code size difference for x86
  - Reduces ia64 by about 500 lines.
  - Maybe in 2.7?

Rusty's Lesson #7

Even If The Code Is Useless, I Learned Something.

But...

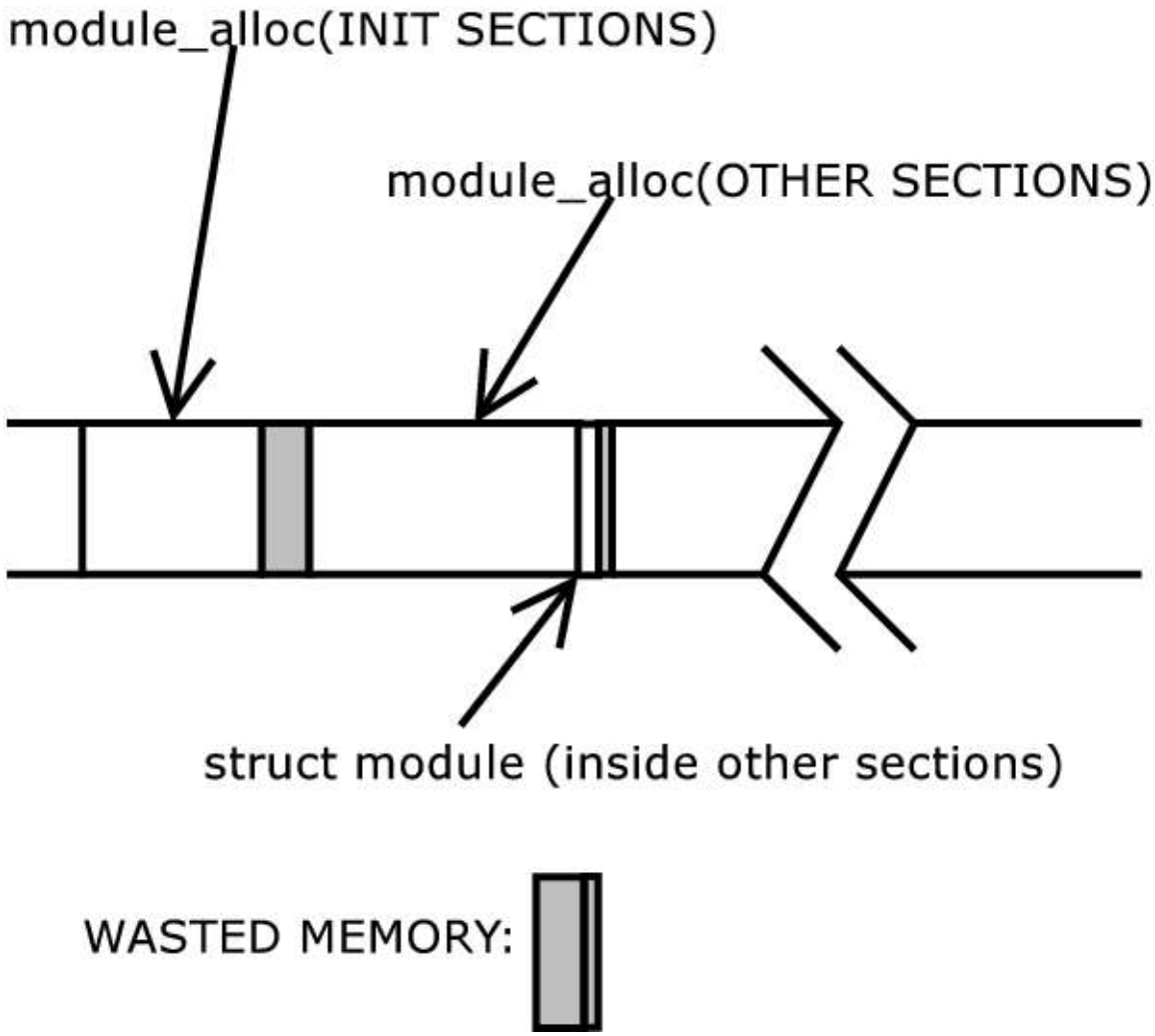
- Richard's Implementation used a neat trick to put the “struct module” inside the module itself:
  - struct module __this_module __attribute__((section(".gnu.linkonce.this_module")))
- This statement inside the module.h header meant that all modules contain the structure.
- The “.gnu.linkonce” means that duplicates are discarded.
- This makes the code slightly neater.
- On December 27th, 2002, this change went to Linus.

Rusty's Lesson #8

You Can Look Really Smart By Copying Ideas From Smart People.

But...

- This also gives Dave Miller his page back:



Rusty's Lesson #9

Be Vicious With Code.
Be Nice With People.

Stability

- It took until January before most modules were back to normal.
  - 20 architectures
  - 1600 modules (some very, very old)
  - Thousands of new users
- Incremental improvements continue still.

Rusty's Lesson #10

No Feature Is Complete Until It Has Lots of Users.

New Features

New MODVERSIONS

- Normally, a module should not be inserted into kernels different from the one it was compiled for
- CONFIG_MODVERSIONS tries to fix it
  - Old implementation worked, but was very messy.
  - Changed names of all kernel symbols exported to modules, using #define
- Kai Germaschewski and I wrote a new one
  - Versions kept in separate sections
  - Versions can be forced with modprobe –force
  - Versioned modules can be inserted into non-versioned kernels

Vermagic

- Less complete that modversions.
- A special string in the ".modinfo" section placed in all modules.
  - eg. 2.6.0-test6-bk1 SMP PENTIUMII gcc-3.2
- If it doesn't match, module will not load
  - For modversions, skips first part of strings.
- Can be forced with modprobe –force.

New rmmod Options

- rmmod –force allows you to remove modules which are in use
  - Great for kernel development
- rmmod –wait allows you to shut down modules which are in use
  - The rmmod will finish when the module usage falls to 0.

Faster Reference Counting

- Per-cpu lockless reference counting for modules.
  - Fastest reference counters in the kernel.
- System to stop all CPUs to examine and unload module
  - Could be used for other things, such as resizing network hash tables.

Per-CPU Variable Support

- In 2.6 I introduced per-cpu variables
  - DEFINE_PER_CPU(type, name)
- Variables are placed in a special section, which is duplicated for every CPU at boot.
- We allocate extra space in this percpu section at boot
  - Modules with ".data.percpu" sections use this space.

Kernel Size

- New Features:
  - New CONFIG_MODVERSIONS: 104 lines
  - module notifiers: 27 lines
  - CONFIG_KALLSYMS: 172 lines
  - Centralized exception table code: 37 lines
  - vermagic code: 15 lines
  - rmmod –force and --wait: 65 lines
  - Fast module reference counts: 148 lines
  - per-cpu module allocator: 179 lines
  - Discarding init sections of modules: 48 lines
- kernel/module.c in 2.6.0-test6: 1952 lines.

Rusty's Lesson #11

Good Code Lowers The Barriers:
More People Can Do More Cool Things.

Conclusion

- The new module code can be changed without having to upgrade userspace tools.
- The new module code is much easier to read and understand.
- All these new features were introduced without breaking the userspace tools.
- Things will keep changing and improving.

Legal Stuff

This work represents the view of the author and does not necessarily represent the view of IBM.

Linux is a registered trademark of Linus Torvalds.

Other company, product, and service names may be trademarks or service marks of others.