Tips for Linux Kernel Development

Jessica Yu, Red Hat

UC Santa Cruz CMPS107

March 7, 2017

Overview

- How do I get started?
- Personal Experiences
- Tips for speeding up compilation
- Debugging tutorial with QEMU

How do I get started?

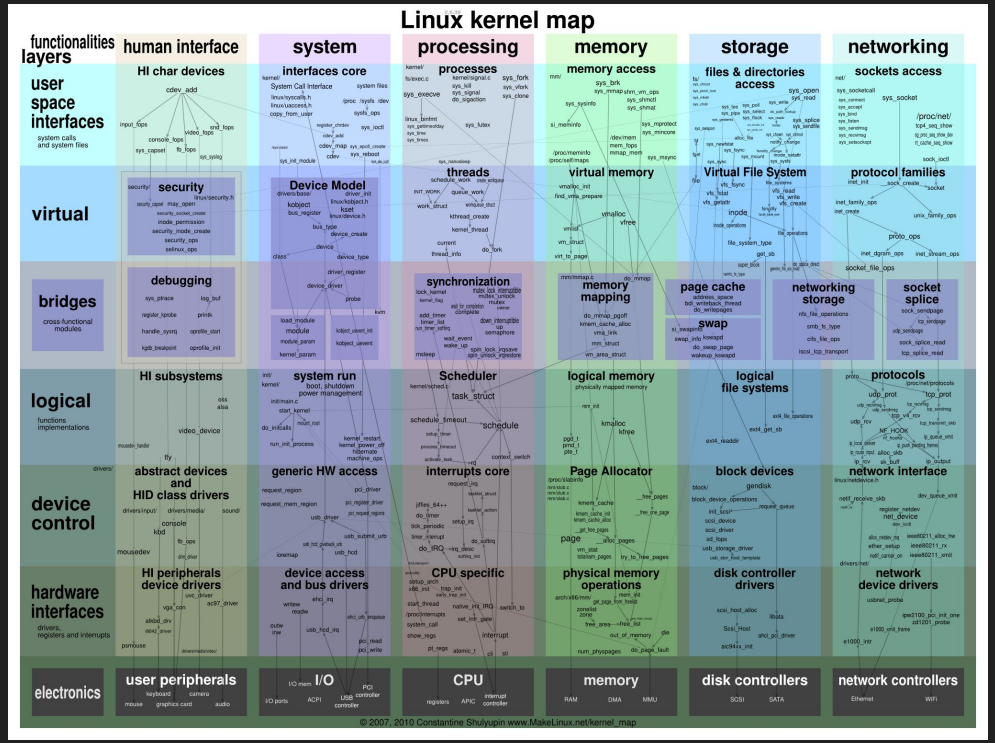


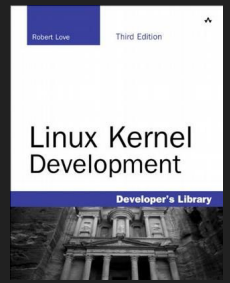
Image source: http://www.makelinux.net/kernel_map/LKM3_2048.png

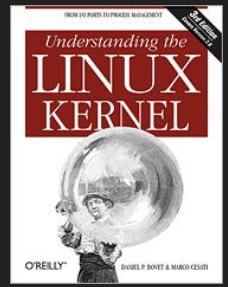
How do I get started?

- Compile your own kernel and boot it! (in a VM, on your laptop, etc)
  - If you boot it on your own machine, make sure to have backup kernels :-)
- Stick a print statement somewhere in the kernel source
  - (in a function you know will run), compile, boot, and look for it in `dmesg`
- Write a trivial kernel module and load it (~15 lines)
- The Eudyptula Challenge
- Linux kernel internships for university students
  - Google Summer of Code
    - TONS of open source projects! Mozilla, FreeBSD, Linux Foundation, Coreboot...
  - Outreachy (formerly called Outreach Program for Women)
    - Similar to GSoC, contribute to open source projects (incl. Linux kernel!)
    - Also open to non-students
  - **Both GSoC and Outreachy are still accepting applications!!**

How do I get started?

- Find a bug and fix it
  - http://kernelnewbies.org/FoundBug
  - https://bugzilla.kernel.org/
- Lurk on various mailing lists
  - Get a sense of how patches are accepted, how to respond to patch comments, etc.
  - All the vger mailing lists: http://vger.kernel.org/vger-lists.html
  - E.g. linux-ext4, live-patching, linux-doc
- LWN.net is a great resource!
- Books
  - Linux Kernel Development (Robert Love)
  - Understanding the Linux Kernel (Daniel Bovet, Marco Cesati)



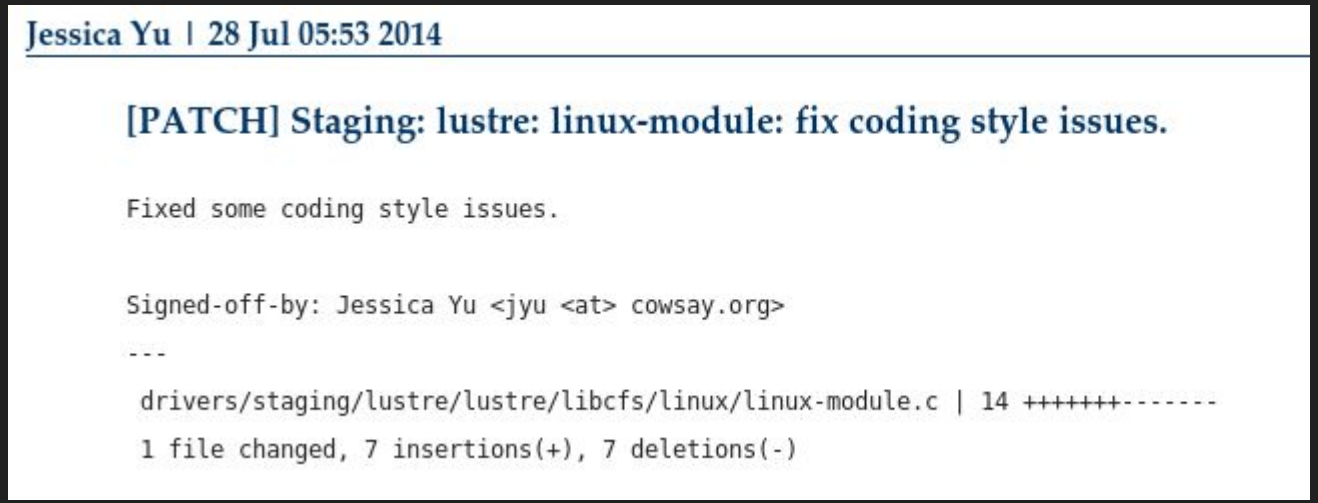


The Eudyptula Challenge

- A series of tasks related to the Linux kernel, geared towards familiarizing you with Linux kernel development
  - starts out easy...gets more complex
    - write a simple kernel module
    - write a syscall
    - send in a patch fixing coding style
    - etc...
  - 20 tasks in total
  - I hear there's a secret prize at the end!

First attempt

- Task 10 of the Eudyptula Challenge.....send a real patch!
  - "Create a patch that fixes one coding style problem in any of the files in drivers/staging/"



Jessica Yu | 28 Jul 05:53 2014

[PATCH] Staging: lustre: linux-module: fix coding style issues.

```
Fixed some coding style issues.

Signed-off-by: Jessica Yu <jyu <at> cowsay.org>
---
 drivers/staging/lustre/lustre/libcfs/linux/linux-module.c | 14 +++++++-------
 1 file changed, 7 insertions(+), 7 deletions(-)
```

First attempt

Greg Kroah-Hartman | 27 Jul 22:01 2014

Re: [PATCH] Staging: lustre: linux-module: fix coding style issues.

On Sun, Jul 27, 2014 at 08:53:50PM -0700, Jessica Yu wrote:

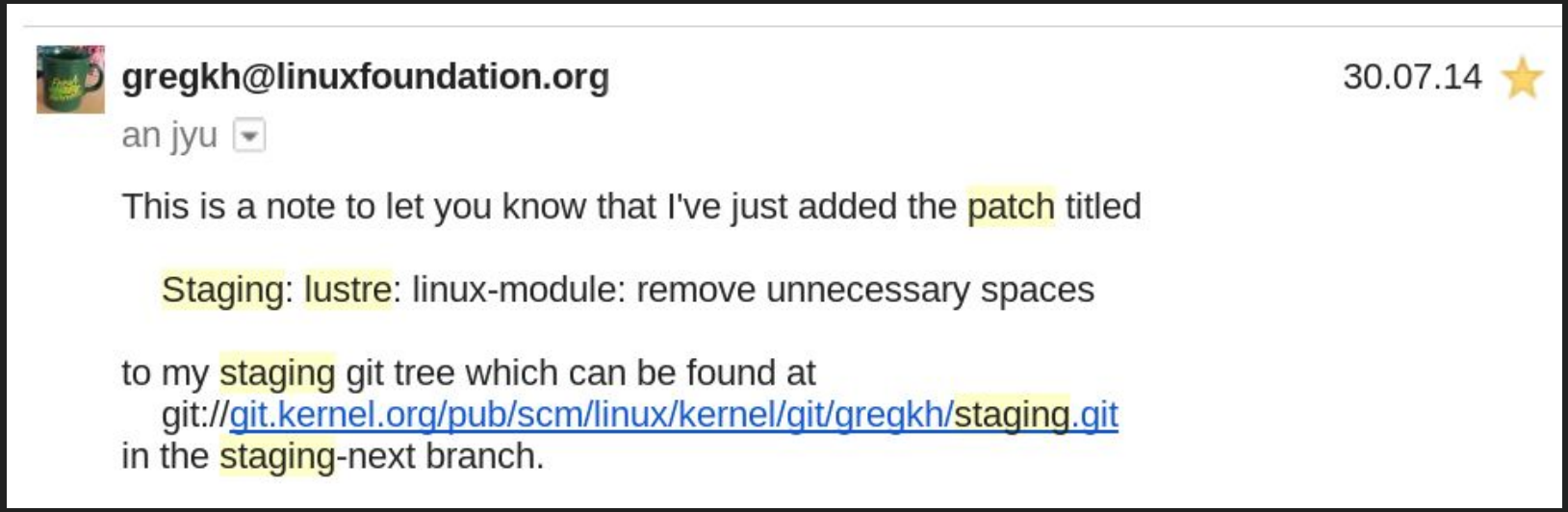
> Fixed some coding style issues.

What coding style issues? Be specific, and explicit.

Care to try it again?

greg k-h

Take two



gregkh@linuxfoundation.org 30.07.14

an jyu

This is a note to let you know that I've just added the patch titled

Staging: lustre: linux-module: remove unnecessary spaces

to my staging git tree which can be found at
git://git.kernel.org/pub/scm/linux/kernel/git/gregkh/staging.git
in the staging-next branch.

Hurray!!!

Tips for speeding up compilation

Tips for speeding up compilation

- **ccache**
  - cache compiled objects
  - greatly speeds up recompilation
- Parallel compilation
  - `make -j 16` on a system with 8 cores
  - Good rule of thumb: `make -j $(number of cores)`
- In-memory compilation (tmpfs)
  - Avoids filesystem syncing/flushing
  - Recommended only if you have enough RAM to spare...
    - Beware of OOM situations
    - `` `free -h` `` tells you total/used/free memory
- All this combined
  - `make -j 16 O=/tmp/linux  271,09s user 195,99s system 550% cpu` **`1:24,85 total`**
  - (Intel Core i7-4800MQ CPU @ 2.70GHz and 16GB of DDR3 RAM)

Tips for speeding up compilation

- Minimize kernel config
  - A lot of time is spent compiling kernel modules!
  - Avoid compiling modules you know you don't need
  - Kernels provided by distributions (i.e. Ubuntu, etc.) tend to be more generalized
    - (provide the config that will work on most machines)
    - means a lot of modules are included
  - `` `make localmodconfig` `` can help slim down your .config
    - uses lsmod to create a config with only the currently loaded modules enabled

Tips for debugging

- Demo time!
  - QEMU + gdb

Debugging with QEMU

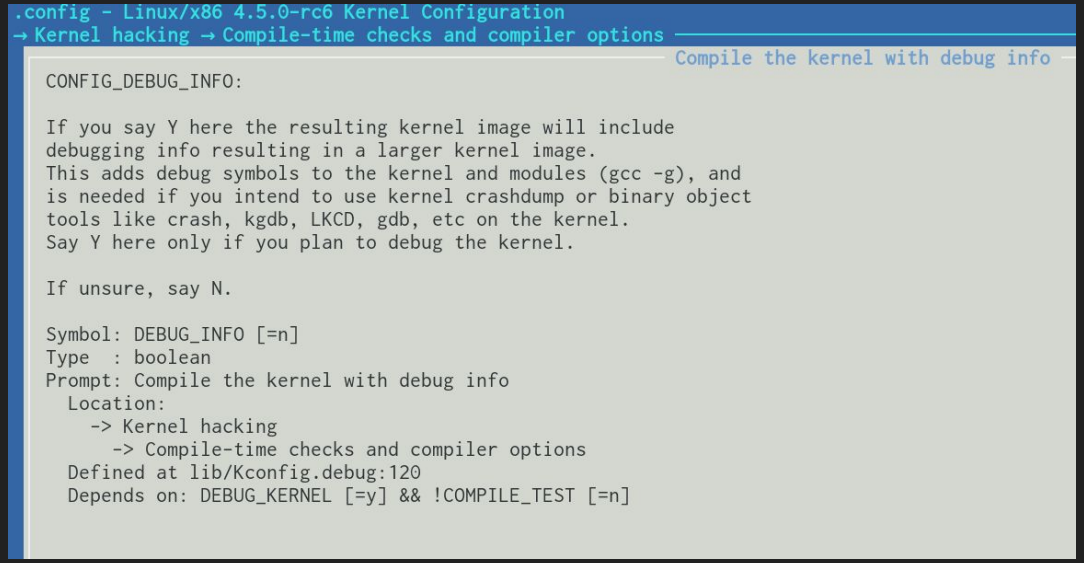
- What is QEMU?
  - A virtualization tool like VirtualBox
  - allows you to run a complete operating system on your machine.
  - very useful for trying out different OS's, testing software, and running applications that won't run on your machine's native platform.
- Get QEMU: http://wiki.qemu.org/Download
  - Your distribution more than likely already provides a package for qemu
- We can use QEMU and gdb to help us debug kernels

Debugging with QEMU

- Compiling a kernel for QEMU
  - `make olddefconfig`
    - `` `make help` `` to see all config targets
  - enable `CONFIG_DEBUG_INFO`
  - `make -j ...`

Debugging with QEMU

- Prepare your kernel for gdb debugging...
  - Turn on debug info in your .config => CONFIG_DEBUG_INFO



Debugging with QEMU

```
$ qemu-system-x86_64 -s -S \  (1)
    -kernel /path/to/vmlinuz \ (2)
    -initrd initramfs.cpio.gz \ (3)
     -nographic \ (4)
     -append "console=ttyS0" (5)
```

1. **-s** Shorthand for -gdb tcp::1234, i.e. QEMU listen on TCP port 1234 for a connection by gdb.
   -S Do not start CPU at startup. QEMU will start as if you set a breakpoint at time zero, and waits for gdb to connect
2. Path to kernel
3. Path to initramfs (we'll just be booting into a very minimal userland, using BusyBox)

Debugging with QEMU

```
$ qemu-system-x86_64 -s -S \  (1)
    -kernel /path/to/vmlinuz \ (2)
    -initrd initramfs.cpio.gz \ (3)
    -nographic \ (4)
    -append "console=ttyS0" (5)
```

4. -**nographic -append "console=ttyS0"**
   a. Disable graphical output so that QEMU is a simple command line application.
   b. All kernel output will go to the ttyS0 serial console, with will be printed to stdio (your terminal).

Debugging with QEMU

- Attaching gdb to QEMU
  - `$ cgdb -ex "target remote localhost:1234" vmlinux`
    - cgdb is just the curses interface to gdb. It's prettier. :-)
    - **-ex** Execute specified gdb command.
    - **target remote localhost:1234** Establish a tcp connection to port 1234 on host `localhost' (where the QEMU gdb stub is listening!)

Debugging with QEMU

- Kernel debugging demo
  - set breakpoints at two functions
    - `cmdline_proc_show()`
      - `fs/proc/cmdline.c`
      - This function is called when we read from `/proc/cmdline`
    - `meminfo_proc_show()`
      - `fs/proc/meminfo.c`
      - This function is called when we read from `/proc/meminfo`

Debugging with QEMU

- Debugging NULL dereference demo

```
[    7.186020] BUG: unable to handle kernel NULL pointer dereference at (null)
[    7.186706] IP: cmdline_proc_show+0x17/0x30   <--
...
[    7.187005] Oops: 0002 [#1] SMP
[    7.187005] Modules linked in:
[    7.187005] CPU: 0 PID: 111 Comm: cat Not tainted 4.10.1+ #10
[    7.187005] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS
1.8.2-20150714_191134- 04/01/2014
[    7.187005] task: ffff8800065d0000 task.stack: ffffc90000254000
[    7.187005] RIP: 0010:cmdline_proc_show+0x17/0x30
```

Then in gdb...

```
(gdb) list *(cmdline_proc_show+0x17)
0xffffffff812de257 is in cmdline_proc_show (fs/proc/cmdline.c:10).
...
```

Debugging with QEMU

- We were able to debug and step through kernel code
- What about linux kernel modules?
  - trickier, but doable
- Modules are **dynamically** loaded
  - gdb won't know where the module will be loaded
  - tell gdb about the module after module load
    - `add-symbol-file` gdb command
      - requires section addresses for:
        - `.text`
        - `.data`
        - `.bss`
      - A module's section addresses can be found under `/sys/module/<module_name>/sections`

Debugging with QEMU

- gdb won't know about your module's symbols unless you gdb about it
- use gdb command **`add-symbol-file`** to give gdb additional symbol information for debugging

```
(gdb) help add-symbol-file
Load symbols from FILE, assuming FILE has been dynamically loaded.
Usage: add-symbol-file FILE ADDR [-s <SECT> <SECT_ADDR> ...]
ADDR is the starting address of the file's text.
(gdb) add-symbol-file /path/to/module.ko <text_addr> -s .data <data_addr> -s
.bss <bss_addr>
```

- Section addresses are found under `/sys/kernel/<module>/sections`

```
$ cat /sys/module/cmps107/sections/.text
0xffffffffa0000000
$ cat /sys/module/cmps107/sections/.data
0xffffffffa00000c0
```

Debugging with QEMU

- Sample module: cmps107.ko
  - Demo code: https://github.com/flaming-toast/cmps107-demo
- cmps107 module does the following:
  - Creates subdirectory "cmps107" under /sys/kernel/
  - Creates file "foo" under /sys/kernel/cmps107/
- The functions we want to step through in gdb:
  - `foo_show()` - called when file foo is read from
  - `foo_store()` - called when file foo is written to

Debugging with QEMU

- Debugging kernel module demo

Debugging with QEMU

- More useful walkthroughs:
  - http://files.meetup.com/1590495/debugging-with-qemu.pdf
  - http://mgalgs.github.io/2015/05/16/how-to-build-a-custom-linux-kernel-for-qemu-2015-edition.html
  - http://wiki.osdev.org/QEMU

Other tips

- Navigating the kernel source
  - The Linux Cross Reference (LXR) - http://lxr.free-electrons.com/
    - Very useful for finding definitions of functions, structs, where a function is called, etc
  - **ctags** with vim
    - jump to function or struct definitions
  - More tips here: http://kernelnewbies.org/FAQ/CodeBrowsing
- Submitting your first patch to lkml
  - Greg Kroah-Hartman has an excellent tutorial on this:
    - https://www.youtube.com/watch?v=LLBrBBImJt4

Google Summer of Code

- Google Summer of code https://summerofcode.withgoogle.com/
  - Applications **open** on March 20, 2017, and **close** on April 3, 2017
  - List of participating organizations this summer: https://summerofcode.withgoogle.com/organizations/
  - **The Linux Foundation**'s GSoC page for this year (list of project ideas for students): https://wiki.linuxfoundation.org/gsoc/google-summer-code-2017
  - Participating GSoC organizations related to **operating systems**: https://summerofcode.withgoogle.com/organizations/?sp-category=operating_systems

Red Hat summer internships

- Program runs from mid-May to mid-August
  - Opportunities to work on the kernel as an intern
  - List of engineering internship positions open:
  - https://careers-redhat.icims.com/jobs/search?ss=1&searchKeyword=internship&searchCategory=17505
  - Drop me an email with resume if you're interested!

Thanks!

Got questions? Feel free to drop me an email!

jeyu at redhat.com

More useful links!

- **Navigating the kernel source**
  - The Linux Cross Reference (LXR) - http://lxr.free-electrons.com/
  - http://kernelnewbies.org/FAQ/CodeBrowsing
- **Submitting your first patch to lkml**
  - Greg Kroah-Hartman's tutorial: https://www.youtube.com/watch?v=LLBrBBImJt4
- **QEMU tutorials**
  - http://mgalgs.github.io/2015/05/16/how-to-build-a-custom-linux-kernel-for-qemu-2015-edition.html
  - http://files.meetup.com/1590495/debugging-with-qemu.pdf
- **Filesystem images you can use with QEMU:** http://fs.devloop.org.uk/
- **Debugging your kernel**
  - https://wiki.ubuntu.com/Kernel/KernelDebuggingTricks
  - http://wiki.osdev.org/Kernel_Debugging
- **Debugging linux kernel modules**
  - https://www.linux.com/learn/kernel-newbie-corner-kernel-and-module-debugging-gdb
- **Participating in the Linux kernel community**
  - http://www.linuxfoundation.org/content/how-participate-linux-community
- **Creating a custom initramfs:** https://wiki.gentoo.org/wiki/Custom_Initramfs