Hacking the Linux Kernel for Fun and Profit

James Bottomley

Hansen Partnership, Inc.

5 April 2008

Introduction

- The Linux Kernel is becoming an increasingly complex place
  - The number of "core subsystem" maintainers is growing
  - The number of supported features is growing
  - The rate of change of code is also (currently) growing
- Often difficult to understand what you're changing.
- Even more difficult to work out what the correct way to change it is.
- However, the kernel has a basic need for talented and motivated contributors

Agenda

1. OK, I understand the Fun Part, where's the Profit?
2. Why you should contribute code to the Kernel (and why your Employer should pay you to do it).
3. Why the Kernel needs you to contribute.
4. Why it isn't as simple as it sounds
5. Where to go to get help

Where's the Money

1. Current total investment in Linux up to 2008: $2BN
2. Annual Income derived from Linux in 2007: $2BN
3. So, there's a lot of money floating around.
4. The problem is, not much of it goes directly to kernel developers.
5. Open Source is about Code not Business Models
6. However Good Business models make money off Open Source
7. Moral: If you want to get really rich, start a company.

Indirect Economic Benefits

- Most open source projects hire the top contributors in their area.
- Even if that's not you, a large number of recruiters use open source mailing lists as a tool.
- Everything you do in open source is on show and easily searchable
  - Recruiters know this.
  - it is also archived forever...

Other Reasons to Contribute

- Direct contributions:
  - There's a bug or missing feature and it's affecting you personally
  - There's a bug or missing feature and it's affecting your employer
  - You (or your employer) has a new feature/driver
- Indirect contributions
  - You have an area in the kernel that you want to work on.
  - You want your employer to sponsor your work on it.

Alternatives (and misconceptions)

- My Product only supports RedHat, SUSE etc. Linux Distributions, so I only need to patch their distribution.
- Distributions are commercially motivated so they're much easier to deal with than Linux Kernel Developers.
- The Distributions are a direct channel to the users, so they're the obvious place to start.
- I can just patch the kernel and ship it myself.

Upstream First Policy

- Major distributions have agreed not to incorporate features or drivers unless they are on "upstream track" for the vanilla Linux Kernel
  - Obviously there's some flexibility in interpretation of this for their best customers
- Primary reason is that it keeps the distribution kernel code and the vanilla kernel code as close as possible, so
  - Maintenance is reduced: the distro can file a bug with the upstream maintainer if there's a problem.
  - Testing is enhanced: users of all distributions are testing the same code
  - Code Review burden is greatly reduced: Can rely on upstream maintainers to review and accept.

What is "Upstream Track"?

- In the vanilla Kernel (Linus Tree)
- In Andrew Morton's `-mm` tree or `linux-next`
  - With the proviso that Andrew has accepted it for onward transmission to Linus.
  - Not everything in `-mm` is designated for onward transmission.
- In a Subsystem Maintainer Tree.
  - Again, it must be designated for onward transmission.
  - Policy on this varies from subsystem to subsystem
- Interpretation within gift of Distribution

Why the Kernel Needs you to Contribute

- The Linux Kernel Code base is incredibly complex.
- No-one understands it all fully
- It maintains its forward momentum and "buzz" because of innovative advances contributed by individuals.
- The more experts the kernel has contributing and assessing the contributions of others, the better it becomes.
- Maintaining the flow of innovation requires a constant stream of fresh talent.

Contributing To The Kernel

- Know where to start
  - Look in the `MAINTAINERS` file
  - Find your driver, or subsystem and see if it has a mailing list.
  - if it doesn't, you have to begin on the Linux kernel mailing list
    * `linux-kernel@vger.kernel.org`
    * very high volume
    * Slightly lower signal to noise ratio.
- Begin by reading the mailing list **not** by coding.
  - Get a sense of where the code is going and what might be acceptable.
  - Read previous acceptances and rejections.

Your First Contribution

- First, make sure you've lurked on the email list for a while to get the feel of the subsystem and the patches.
- Then, your initial patch should be small, just to get the feel of the process
  - Find a tiny bug or misfeature and fix it.
  - Will give others confidence in trusting you.
  - Will get you used to the patch submission process
- If all goes well, and you think you understand how the subsystem is working, then you can begin your big driver/feature.

Rules for Coding your Feature/Driver

- Release Early, release often
  - Your first patch, doesn't even need to be a patch, just a "this is how I'm thinking of coding this" email.
  - Makes sure you're going in the right direction
  - Gets feedback (and buy in) from others in the development
  - Allows any corrections to be made easily (before you've coded another 10,000 lines of code dependent on the piece that the maintainer wants changed)

Accepting Feedback

- Pay attention to feedback on your code
  - Even if you know your own driver/feature, others probably know the kernel better.
  - Even in your own code, another pair of eyes may spot a bug you missed.
- Some feedback is more valuable that others
  - Every mailing list has its share of armchair coders.
  - If you studied the list first, you should have a pretty good idea who they are.
  - Can also tell by what type of reply from others the feedback elicits.

Section Mismatches

Unless they break the build, or if there currently are 0 and they make it non-zero, people seem not to care....sad. Probably the same for sparse/checkpatch, "there's plenty already, I can't be bothered to look"

Re: Section Mismatches

> Unless they break the build, or if there currently are
> 0 and they make it non-zero, people seem not to
> care....sad. Probably the same for sparse/checkpatch,
> "there's plenty already, I can't be bothered to look"

checkpatch does not parse C, it uses heuristical regexes.

That makes it very different from sparse or the section mismatch finder which do not output false positives.

Re: Re: Section Mismatches

> > Unless they break the build, or if there currently are
> > 0 and they make it non-zero, people seem not to
> > care....sad. Probably the same for sparse/checkpatch,
> > "there's plenty already, I can't be bothered to look"

> checkpatch does not parse C, it uses heuristical regexes.
>
> That makes it very different from sparse or the section
> mismatch finder which do not output false positives.

Even by the exalted standards of LKML which sometimes
seems to make a virtue of misinformation, four wrong
statements in twenty seven words is pretty impressive
... I salute you!

Why Contributions Usually Fail

- One of the most classic is Coding Style
  - Read the kernel coding style document `Documentation/CodingStyle` and follow it.
  - Not conforming really does matter, because it makes your contribution harder to follow and more difficult to maintain.
  - This really, **really** does matter, so people will be anal about it.
  - Redoing the style is fairly easy and, hey, if that's all they complain about, they must have liked the code ...

A success story: the initio fiasco

- 21 May 2007: Alan Cox redoes the entirety of the initio driver
  - he does get a tester to make sure it works
- Over the months, several interfaces have been changed and updated
- 17 Dec 2007 A user posts a bug report saying basically that the initio driver no longer works in recent kernels
- Oops.
- Long discussion on mailing list
  - Alan's tester isn't there anymore
  - No-one really knows what's wrong with the driver

From: Stuart Swales <stuart.swales@croftnuisk.co.uk>
To: linux-scsi@vger.kernel.org
Subject: [PATCH 2.6.24-rc8-git6] initio module hangs on loading fix set
Date: Wed, 23 Jan 2008 20:00:48 +0000 (14:00 CST)

I've verified (on my Initio 9100 with a DAT drive) that the 2.6.24-rc8-git6 initio module still hangs on loading.

These fixes (other than the printk) are needed to get the module to load ok (and work correctly) with my adapter & tape drive.

Where to Contribute

- Kernel is divided into "Subsystems"
- 50% of the kernel code is in `drivers/`
  - That's over four million lines
  - 90% of the kernel bugs are in drivers
  - Especially `drivers/scsi`
- Rest is architectures (`arch/`) core kernel (`kernel/`) Memory Management (`mm/`) Filesystems (`fs/`) and networking (`net/`)
- Oh, and `Documentation/`

What is The Kernel API?

- Simple answer: "There isn't one"
- More complex one is that there is, it just isn't stable
- Good reason for not stabilising it
  - Allows faster innovation
  - A changing API lets us correct it when we get it wrong
- Several books you can read, but best source is the kernel itself
- If you don't like what you find, think about documenting it better ...

Getting The Code

- On any distribution install git
- `apt-get install git`
- `yum install git`
- ...
- Download the Kernel
- `git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git`

Mailing Lists

- `linux-kernel@vger.kernel.org`
- `linux-fsdevel@vger.kernel.org`
- `linux-scsi@vger.kernel.org`
- ...
- Vger is a Majordomo system. To get a full set, just send a message with 'help' in the body to `majordomo@vger.kernel.org` to get started.

Websites

- `www.kernelnewbies.org` — best place to get started; packed with information
- `www.kernel.org` — repository of most actual kernel code; more expertise required in the interface
- `marc.info` — archive of all linux related mailing lists (and quite a few others).

Conclusions

- Submitting patches is different from any other industrial process you'll have been through before
- The trick is to understand the constituency you're trying to convince to accept your patches.
  - i.e. study the mailing list
- Release early and release often.
- The Kernel API is *huge*; pick a small part of it to begin with.