

# **Linux Device Driver Roadmap Or Why we Don't Have One**

**James Bottomley  
SteelEye Technology**

**9 November 2006**

## Introduction

- Business Product decisions are made based on a variety of factors, including:
  - Product Availability
  - Market Conditions
  - Consumer Delivery
- In general, the only one a company fully controls (it hopes) is Product Availability.
- Other factors depend on Commodities:
  - Common Off The Shelf (COTS) hardware is now Commodity
  - Increasingly, the Operating Systems is becoming Commodity.

## Introduction (2)

- So Market Conditions and Consumer Delivery are often dependent on Operating System (and Hardware) features.
- Knowing when these features will arrive with good accuracy is critical to making the correct business product decisions.
- Thus, the basic input needs to be the Roadmap.

# Agenda

1. Examine why people want a roadmap in the first place
2. Introduce the Open Source Vision guided but Technically Led development model
3. Compare and Contrast.
4. Common Myths and Misperceptions.
5. Case study: Linux Storage
6. Influencing the Linux Technical Roadmap

## Traditional Roadmaps

- A traditional roadmap is simple; it's usually a linear chart with a list of features ordered by the date it's proposed they will be realised.
- Rarely do people simply want to view a roadmap
- Usually they want to see it so they can influence it to include whatever set of features they want in whatever time frame.
- Thus, also need to study the methods of *influencing* traditional roadmaps.

## Influencing Roadmaps

- Direct: “I’d like to talk to someone in charge to explain my requirements.”
- Indirect: Open an Enhancement Request or Bug.
- Oblique: Talks to a business partner, who has much more influence with the roadmap about getting your feature in.
- Tangential: Get your feature into an approved or upcoming standard and then demand to know when the standard will be implemented.

# The Standards Game

- Generally played for three reasons
  1. You have a feature you want everyone to adopt
  2. Somebody else has a feature you don't, so you join the standards body to disrupt it.
  3. Everyone else is ahead of you in the standards game so you join to delay.
- Takes quite a lot of time and effort to play
  - Usually only large companies

## Using These Methods in Linux

- Direct: Well ... you can try Linus, but ...
- Indirect: We have a bug database ([bugzilla.kernel.org](http://bugzilla.kernel.org)) but don't try putting a feature request in it.
- Oblique: Some companies do have influence ... but they won't be able to help
- Tangential: May work ... some standards (POSIX, SAM3) are implemented (well, partially) ... but just because a standard exists doesn't mean we'll implement it.

# The Open Source Model

- Vision Guided
  - If you have a demonstrably good idea, you might be able to persuade others to implement it.
  - Van Jacobsen—Network Channels.
  - However, usually people advocating some new vision are given short shrift on the mailings lists.
- Otherwise, kernel is technically lead
  - This means the person interested in the feature provides the code
  - and that the code must conform to open source principles
  - Or, you could pay someone else to do it.

# Industry Problems with the Open Source Model

- Being Code or Vision lead means the entity requesting the feature must know how to implement it
  - This isn't usually true.
  - Consumers don't necessarily have expertise in producing
  - Example: SteelEye and Array ownership for clusters
- Presents a huge impedance mismatch to overcome
  - Lots of companies consult for services like these
  - Or, could look for hidden talent internally.

## Compare and Contrast

- Old model had three players
  - OS Producer
  - OS Consume
  - VAR
- OS Producer makes money by leveraging VARs and thus has a large focus on keeping them happy.
- Linux OS Producer is a volunteer community
  - No requirement to make a profit
  - Thus, no focus on keeping VARs happy

## Compare and Contrast the Economics

- OS Producers often had pay for play VAR programmes
  - often at many tiers (gold, silver, platinum)
  - Each tier has a list of services you can expect for a corresponding annual fee
- Linux levels the OS playing field
  - No pay for play ... everyone may contribute
  - Only, to contribute you must be capable of doing it
  - Cultivating in-house kernel expertise costs money
- Thus, costs of old and new models are usually a wash
  - CFO doesn't expect this (Linux is supposed to be "free"); probably has already absorbed the VAR programme budget elsewhere.

## The Bottom Line

- VARs *require* kernel coders to look after their interests.
- Once this is realized, the question is where to get them from.
- Hiring, the standard option, is rendered difficult
  - Pool of available talent is small
  - Other companies (like Google) are driving the price up
- So, finding home grown talent is going to be by far the easiest course.
- So start cultivating it *now* ...

## Myths and Misperceptions

1. Someone else will implement the code for us
2. We already have code for \$OtherOS, we can just open source that and then someone will pick it up for Linux
3. We coded up this feature, here it is, put it in the kernel
4. If I get my code accepted into the kernel I no longer need to maintain it
5. If I maintain my code in the kernel then no-one else can touch it

## Case Study: Linux Storage

- Will cover two separate issues in this case study
  1. How did SteelEye get into Kernel Coding (as an application VAR)
    - and, more importantly, why do we continue to subsidise maintenance of the SCSI subsystem.
  2. What does the Linux Storage Roadmap actually look like
    - In so far as a roadmap exists
    - and, obviously, it's in terms of technology not features ...

## About SteelEye

- Founded in 1999
- mission to bring application HA to Linux
- Achieved by buying and porting the NCR LifeKeeper HA Cluster product to Linux.
- Company hired a large swath of NCR engineers for initial staffing
- Most of whom were kernel coders from the NCR UNIX SVR4MP OS called MP-RAS

## LifeKeeper and Linux

- Most porting application based ... not much of a problem
- However, base of LifeKeeper was shared storage clusters; two problems
  1. Shared Parallel SCSI buses didn't work in Linux in 1999-2000
  2. The storage ownership model (SCSI Reservations) LifeKeeper used in both MP-RAS and Solaris didn't exist in Linux
- Lucky accident: being mostly kernel engineers we can figure out what the problem is and how to correct it in both cases.

## Our Solution

- Found a set of Fixes ... easy
  - Elected to modify Linux to implement reservations at user level
    - \* This, accidentally, nicely aligns with the current kernel philosophy of moving policy to user space
  - Actually modified SCSI mid-layer
  - and aic7xxx driver
- Tried to get them into the kernel via Red Hat ... less so
  - But much easier in those days
  - Actually formed working relationships with Red Hat SCSI engineers

## Storage Ownership—Perhaps not so Accidental

- First tackled problem (Shared SCSI buses) taught us the difficulty of modifying kernel
  - Problem: SCSI so vital, so many interested parties, agreement on code changes hard to achieve.
  - This made us conclude that minimum and most generic changes were the ones most likely to be accepted.
    - \* This principle *still* applies today
- So concluded to comply with this
  - Storage ownership would be mediated at user level
  - with minimal necessary kernel support
  - Although kernel changes were still necessary

## The rest is History

- First changes taken by Red Hat (not vanilla Kernel) for 6.1 (June 2000)
- Next targeted OS were SuSE and TurboLinux.
- Realised that easier to apply changes to vanilla kernel ASAP and wait for all distros to pick them up
  - So, accidentally, we hit on “upstream first” policy
  - pragmatic: ease engineering support burden
- In 2003 became SCSI Maintainer
  - Shared Storage and Ownership model never broke again ...
- in 2006 SteelEye has 3 Linux kernel engineers (two maintainers) on staff.

# The Linux Storage Roadmap

- Totally untraditional format
  - Not a time line
  - Actually a TODO list
  - Engineering features, not end user features.
- Presented at various dates
  - Mostly kernel summit in 2002, 2003 etc
  - More sorted out at Vancouver Storage Summit 2006.
  - Also on the mailing list `linux-scsi@vger.kernel.org`

## The Actual Roamap

- Thin down SCSI Subsystem by
  - Splitting out helper libraries for various transports
    - \* Mostly done as the transport classes
  - Move functionality that should be shared up to block
    - \* Tagging done
    - \* Error handling
    - \* Queue Grouping
    - \* Command Timeout
- Shared ATA ULD for SAS/SATA
- dm-multipath to request based infrastructure
- Barrier support for TCQ

## Influencing the Linux Roadmap

- “It’s All About the Code, Stupid!”
  - The actual Linux roadmap (or TODO list) is determined by the people actually doing it.
    - \* Simply because those coding are those doing the TODO items
    - \* Thus, those adding to the list are really those who’re going to be coding the features anyway.
  - to be one of those, you have to be contributing code
- Thus, the influence of a company on the Linux roadmap is almost directly proportional to the quality (and commitment) of its coders.

## How easy is This?

- As an operating system, like all technology based initiatives, Linux depends for its primacy on its innovation stream
- Requires a constant flow of new ideas and patches
- So we're always on the lookout for new talent
- best reception goes to those who propose the most generic (or most abstract) ideas
  - i.e. don't just code for yourself
  - Code a the way that lets the maximum number of people benefit from your code

## Lateral Observations

- Linux really is a pay for play model as well
  - You just pay with code instead of cash.
- Effect of this expands the requirements for kernel (and other open source project) coders at companies who've never previously needed them.
- Effect is to increase the coder base, and hence the innovation stream into Linux
- Which, in turn drives the kernel on ahead of the competition
- A Virtuous Circle.

## The New Methods of Influence

- Direct: Write your own code
- Indirect: Get someone else to write the code for you
- Oblique: Form a consortium of interested parties to write the code
- Tangential: adapt the code from something else.

## Conclusions

- The traditional mindset for influencing feature development doesn't work in Linux
- Worse, it may actively work against you.
- To play in the new arena you need individuals who can gain respect through code
- However, anyone can play ... regardless of company size.