Trends in Linux Kernel Development

Andrew Morton
<akpm@linux-foundation.org>
<akpm@google.com>
FOSDEM 2007
February 2007

Looking Forward

- What features are being merged into the kernel?
- Who is developing these features, and why?
- The role of professional developers
- The importance of private developers, and how individuals can contribute.

Types of systems which use Linux

- Servers (and other large machines)
  - database, web, file
  - Scientific computing
  - Most funding is in this area: hardware companies, software companies whose customers use Linux
- Desktop
  - Less important to companies than the server
  - But more important to the developers and to their most immediate users
- Consumer
  - Often “small PCs”: DVR, PS3, instrument control, etc
  - Funding for kernel work is lower, affected by the “embedded problem”

Types of systems which use Linux (cont'd)

- Embedded
  - Smallest devices: cellphones, PDAs, network gear, etc.
  - Usually non-x86, often no-MMU. Diskless.
  - There is relatively little funding for embedded development in Linux.

Why companies fund Linux development

- Three main situations
  - Hardware vendors
    - sell hardware on which their customers wish to run Linux
  - Software vendors
    - sell software and services to customers who run Linux
  - Device manufacturers
    - make complete products which include Linux
- Hardware and software vendors' customers expect to upgrade their kernel versions
  - It turns out that the best way in which to provide features to these customers is via the upstream kernel
  - This logic is the source of much of the funding for kernel development
- Some device manufacturers also have upgrade plans, hence they will fund kernel.org development

The “embedded problem”

- Unlike servers and desktop, most embedded devices use a single kernel version for the whole product life
  - Pick a kernel, customise it, ship
- No kernel upgrade is planned, so there is little motivation to merge customisations into upstream
- Timelines and budgets are tight in embedded
- So there is little involvement in kernel.org development from embedded developers
- There is *some* involvement, but often from hardware companies and software/service providers (again)
- Despite all this, we do care about embedded and we work to improve kernel support for it

Technology walkthrough

- What is happening now?
- What is likely to happen soon?
- Who is doing it?
- Why are they doing it?
- There are always surprises

Technologies: server

- Infiniband
- Network protocols, congestion management, etc
- SATA/SCSI evolution
- NUMA evolution
- Virtualisation (KVM, VMWare, lguest, Xen)
- Containerisation
- Resource management
- kexec and kdump
- kprobes and systemtap
- ext4

Technologies: desktop

- Hotpluggability: devices, CPUs, nodes, memory
- Ongoing power management work
- Neverending stream of framebuffer drivers
- Direct-rendering drivers
- Much work ongoing with input, sound, USB, 1394 drivers
- Improvements to memory management, interactivity

Technologies: consumer/embedded

- Much activity in DVB/Video4Linux
- Dynamic ticks, hrtimers (needed by OLPC)
- Ongoing footprint reduction
  - More fine-grained configurability
- Improving NoMMU support
- New architectures (FRV, avr32, blackfin)
- OMAP, SPI
- More features will be merged from Ingo's -rt tree

Instrumentation

- Ongoing need to expose more information about kernel operation for debugging and tuning
  - Probably we're not doing enough of this
- Per-task statistics (taskstats): improved task accounting
- Per-task IO accounting
- Per-process memory footprint monitoring
- Perfmon
  - Access to CPU performance counters
  - More for userspace than kernel
  - Progress is slow

Kernel core

- kevent: efficient unified event delivery
- utrace: rewrite of the ptrace support code
- syslets/fibrils: asynchronous system calls
  - will improve (and obsolete) the existing partial AIO support

Debuggability

- Kernel development is highly decentralised
  - Developers and testers are widely separated
  - Hence Linux needs exceptional remote-debugging ability
- A lot of self-checking code is already in there
- The locking dependency checker was recently merged
- Fault-injection framework was recently merged
- New debugging features are readily accepted
- Maybe one day we'll merge a kernel debugger (I prefer kgdb)
  - But a debugger is for local developers, not for remote debugging

Cleanups

- We merge a lot of cleanup patches
  - Code refactoring
  - Whitespace fixes
  - Replacement infrastructure (eg, mutexes, RCU)
  - API changes (eg: timers, workqueues, PCI API)
    - Followup patches to fully migrate to the new API
- Lots of ongoing churn, and some risk
- But we believe it is important
  - We expect the codebase to be actively developed and maintained for decades to come
  - Improvement to the consistency and overall understandability reduces maintenance cost in the long term
- The kernel has become a lot better as a result

Surprises

- Interesting features are regularly submitted without prior announcements
  - eg: lockdep, kevent, KVM, async-syscalls
- The quality of these submissions is often high
- They often get merged quickly

The role of private contributors

- Probably most kernel work is performed by professionals
- Private contributors are important, especially in desktop-related development
- It is often hard to know if a contributor is professional or private
  - If they're good, they don't stay private for long

The role of testers

- An area where private contributors dominate
- Many testers are individuals who simply want to help the effort
- External testers are a key part of the whole kernel effort
  - A key reason is that the kernel must run on thousands of different types of machines – more than the developers have access to
- The whole kernel project would fail without our testers
- Testing is an easy and valuable way to contribute

How to contribute by testing

- Grab latest -linus snapshot, use it in normal daily activities
  - Once per week or once per month
  - Fedora, openSUSE and probably others provide kernel snapshot packages. Using these is OK.
- Report any problems
  - If they're recent regressions, email is appropriate
    - Try to Cc the developer, and the appropriate list.
    - Also Cc linux-kernel so I get to see the report
  - If it's a longer-term bug, use bugzilla.kernel.org
- If possible, be prepared to help diagnose the bug
- Using git-bisect to identify the buggy patch is ideal