The Kernel hacker generations

Jan 2007

Andi Kleen, SUSE Labs, Novell
ak@suse.de

Linux kernel as a long term project

- Developed by many people over time
  - with various interests and motivations
- Trends not people
- No temporal ordering
  - Overlap
- Bias to newer developments

The janitor generation

- Large codebase needs maintenance
- kernelnewbies / kernel-janitors
- Clean up code base in simple ways
- Generate many changes
- Patch infrastructure handles it now
  - Didn’t use to.
- Graduation to more difficult projects?

The loginname-tree generation

- Starting with famous -aa and -ac trees
  - then became a trend
- Relieved a lot of pressure during the "merging crisis"
- Tests patches not yet in mainline
  - But can so many trees find enough audience?
- Collection of different "branches"
- Mostly replaced by one big tree (-mm)
  - collection of more topical trees
  - and distribution trees of course

Corporate generation

- When Linux became big business ...
- Drivers
- Hooks, hooks, hooks
  - Often steered to better solutions
- Great projects
  - But not always outside the company
  - Missed much useful from the early submissions

Corporate generation II

- Changes usually developed in a closed way
  - and deliver a finished / QAed patch.
- "patch publishing" model quite different
- Submission originally very lossy
  - Lost some useful things early
- Works well now in many cases
  - Introduction of new contributors still needed
  - Review still a problem

The Russian mathematicians

- Not all Russians or even mathematicians
- "... room full of hackers operating under a single name"
- Very bright people
  - Solve tricky problems
- Thankfully we got them as the kernel got harder
  - Especially MP scaling
- But in the end I hope we don’t need them anymore

Flame generation

- Flames always existed
- But tone seems to get nastier
  - Especially during review
- Danger of scaring valuable new people away

Deadline generation

- Linux kernel development used to be relaxed ...
- 2 week merge windows
  - And it’s unpredictable when the window opens
- Creates a lot of time pressure for hackers to get changes in
- Code with (soft) deadlines now

(Developing) tester generation

- Traditionally Linux relies on users as testers
  - No formal QA in kernel.org
- Larger user base doesn’t use bleeding edge kernels anymore
  - Still got good hardware coverage
- More and more complexity that is hard to test casually
- Systematic regression testing
- Internal test code
- Test code that is not often tested

(Slowly developing) bugmaster generation

- When to do a release?
- Depends on the bugs

- Growing bug numbers are (probably) a big problem
  - But we actually don’t know for sure
- Theory:
  - Fact: Source is growing
  - Even if bug rate / source line is constant this means ...
- Keeping track of bugs
  - Widely scattered
  - Distributions versus kernel.org

What does a bugmaster do?

- Work with bug reporters to get basic information
- Prune duplicates
- Weed out dead bugs
- Set proper priorities
- Nag maintainers to fix the bugs
- Keeping track of regressions
- Don’t need to be experts on any kernel areas
- Don’t need to fix the bugs!
- Know what state a release is in
  - and how Linux is doing on the bugginess scale

(Slowly Developing) Technical writer generation

- Complex systems need documentation
- Internal documentation
  - Needs maintenance
- Maintain man pages
- Future: Work on "great unified Linux documentation tree"?

Developing: Destructive generation

- Stress kernels to find bugs that normal testers don’t hit
  - fsx
  - fsfuzzer
- Distributions have some people
  - but they don’t work on mainline
- and some gotten from other OS

More destruction

- Destroy a kernel ...
- ... and then write good a good bug report about it!
- Internal white box testing
  - Inject errors
- Inject errors
  - lockdep
  - malloc failure tester
- More destroyers needed

Future: The reviewer generation

- A Generation I would like to see

- Source code growing quickly
  - Lots of new programms
  - ... and Linux relies on code review to keep code quality high

- Reviewing bootle neck
  - ... especially for "unsexy" code

Reviewer generation II

- Maintainers do a lot of reviewing
  - but they can’t do it all
  - and there is often no clear maintainer
- Interest depends a lot on current hype level
  - and the name of the submitter
- But for others it is hard to get review

Good review

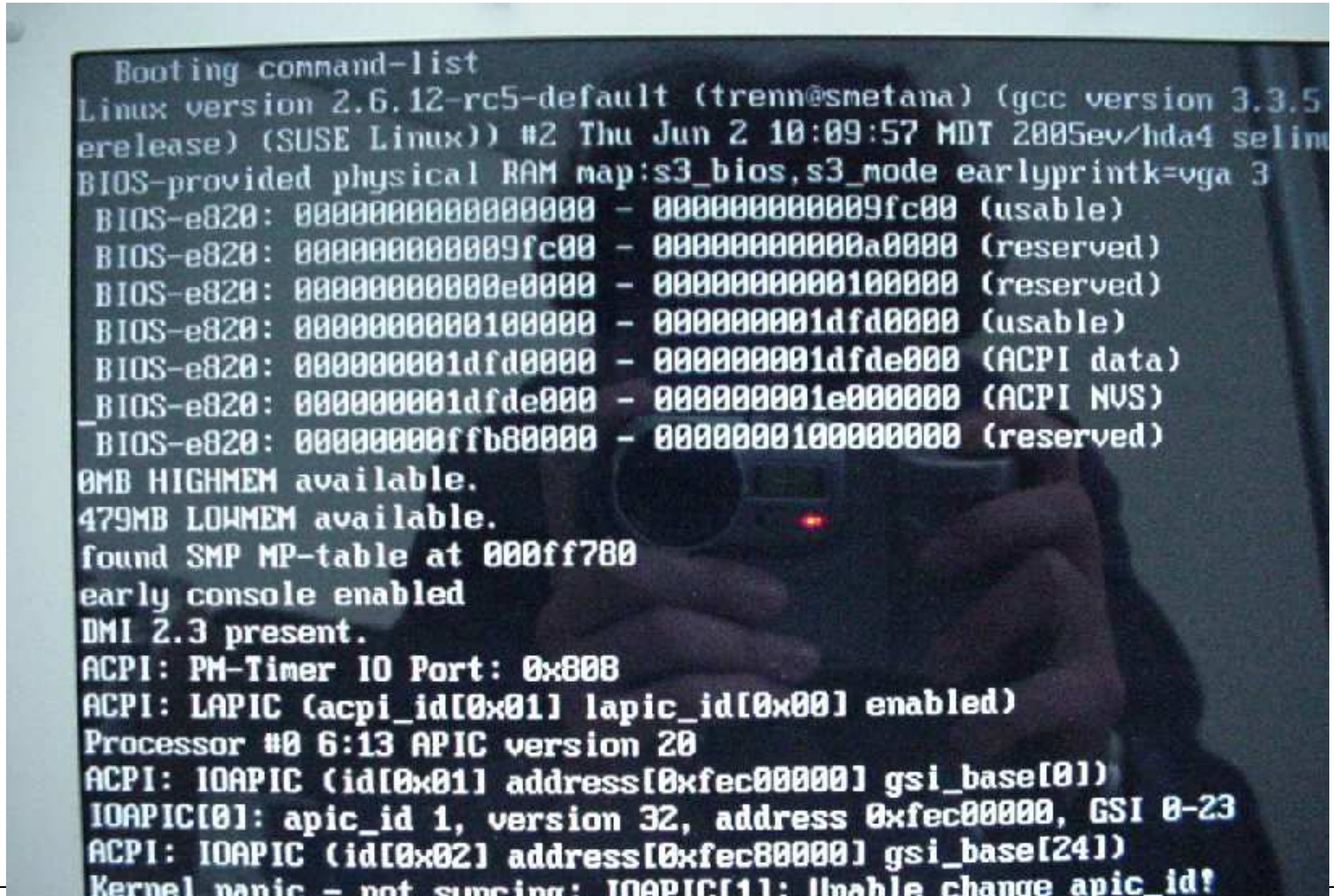
- Coding style is not all

Really good review

- Proper review is a lot of work
  - Maintainers can’t do all the low level review
  - Often there is no clear maintainer

- People who read code well
- and are open minded
- and ask a lot of "stupid" questions
- look for simple logic errors
- recognize bad idioms
- like reading code

Often state of the art in kernel debugging ...



Debugging generation

- From "real men don’t need kernel debuggers" to ..
- ... tens of debugging options
- The tale of the standard debugger
- Crash dumps

Maintainer generation

- Kernel got a "middle management"
  - Or even multiple levels
- (tongue-in-cheek) "... when one looks more at diffstat than patches"
- Spend more time on reviewing / merging / bug triaging etc. than hacking
- Better investment of time than directly hacking?

LocalWords: dups mvc jpg