Polishing the Dirt

Deploying vendor software in embedded Linux systems

Porting legacy RTOS code to Userspace drivers framework

Vendor software?

- Software that comes from semiconductor vendors together with the chip
    - Firmware an chip enablement
    - Basic examples
    - Middleware
        - Intrusion into Android framework
    - ...and even applications
- Ok. Why?

Product development objectives

- Marketing pressure
  - Faster time to market
  - Feature richness
- “Development minimization” paradigm and stereotypes
  - Building from blocks
  - OSS perception: “just take what you need”
- Why not use only OSS solutions then?

OSS for an end user product?

- OSS is not production ready
  - Always in development stage
- Vendors produce specific chips per big customers' demand
  - No OSS support out of the box
    - Needs quite a bit of tweaking
  - Someone is to implement support for that
    - Vendor?
    - End user product manufacturer?
    - Third party?

Vendor software evolution

- “Complete solutions” instead of chip enablement
  - Save development time
  - Mostly integration work
  - Less risk for the product producer
  - Vendors take the responsibility for their code
    - Bugfixing
    - Maintenance
- So why so sad?

Vendor software: as it is

- A lot of redundant code
  - Support for multiple platforms
  - Support for chip families
    - “Sorry, you're not our only customer”
- Many levels of indirection
  - Ex. 5 callback levels
- Legacy code
  - Developed for years, the base might be old
- And that's not it...

Vendor software and Linux

- Origin is very different
  - Initially written for an RTOS
  - Assumes single address space
  - No kernel/userspace separation
- Clean porting to Linux is not a no-brainer
  - Requires deep knowledge
  - Time consuming
- Ends up with a quick-and-dirty porting

Vendor software: pray or deny?

- Make the most of the vendor SW
  - Let the vendor do the work (development/integration)
  - Leave maintenance to the vendor

- Minimize the use of the vendor SW
  - Treat it as a prototype
  - Develop own solution

- Solution analysis before deployment
  - Evaluation agreement

Collaborative approach

- Good as long as you're a customer
  - No local knowledge of the internals
  - No community acceptance
  - You totally depend on the vendor support
  - Each service pack is a problem
- Forward porting may become a problem
  - Another kernel version
  - Changes in the framework

“Denial” approach

- Might be good in the ideal world
- Not applicable in the real one
  - Takes too much effort
  - You won't get complete control anyway
    - Binary parts (firmware)
  - You are not backed up by the vendor
    - e. g. for the firmware upgrade

Analytic approach

- Apparently the best, but...
  - Criteria are unclear
  - Usually requires additional agreement with the vendor
  - Analysis itself takes time
- Still it's usually worth it :-)
- Our proposal: different criteria for
  - Open source vendor software
  - Non-open source vendor software

Open source vendor software

- Mostly kernel-related
  - Device drivers
  - Generic extensions
  - Hackery in generic code
- Criteria are clear enough
  - LDM conformance for drivers
  - No hackery as above
  - Mainline acceptance for generic changes

Non-OSS from vendors

- Proposed criteria
  - Modularity
  - Security
  - Proper kernel/userspace interaction
    - Consider deploying userspace drivers
- NB!
  - Need to understand the reasoning behind “wrong” solutions
  - Need to communicate back to the vendor to not lose the warranty/support

Userspace drivers framework

- Kernel framework for having part of driver functionality in userspace
- Authors/credits
  - Thomas Gleixner, Hans-Juergen Koch
- Meant mostly for simple devices
  - Complete kernel driver might be an overkill
- Also can solve some licensing issues
  - No binary kernel module nonsense
- Userspace IO system (UIO)

UIO: highlights

- Kernel “stub”
  - Low level stuff (interrupt handling)
  - /dev/uioX device files
- Userspace daemon
  - Driver “logic”
  - Interaction with the kernel stub
    - File operations, mmap()'ing etc...

Polishing examples

Vendor OSS example:
WLAN driver stack

- Transport part
  - SDIO: abuses existing OSS driver
  - SPI: doesn't use kernel SPI framework
- WLAN part
  - Initially written for the legacy RTOS
    - Licensing issues
- Consequences
  - Nowhere near community acceptance

Possible improvements

- Use mainline kernel features where possible
  - Use standard SD controller driver
  - Conform to LDM
- Use userspace drivers framework
  - Move WLAN state machine to userspace
  - Keep networking part in kernelspace
  - Keep SDIO part in kernelspace

Improvements: impact

- Stock SD driver deployment
  - SD card reading on resume problem is gone
  - Less code to maintain
- WLAN SM to userspace
  - No licensing issues
  - Some speed increase
    - Faster interaction with wpa_supplicant

Vendor non-OSS example:
FM radio

- Solution completely in userspace
  - Uses AF_BLUETOOTH socket to
    - Send commands to the chip
    - poll for events from it
  - Needs BlueZ hciattach to be running
  - Only can read all events at once
    - Can only process 1 command at a time
    - Has to ignore events while waiting for command completion
    - High latencies in event processing

Possible improvements

- Implement a custom driver interacting with BT UART
  - No need to take HCI interface up
- Implement event processing as an userspace driver
  - Event reading in kernelspace
  - FM state machine in userspace

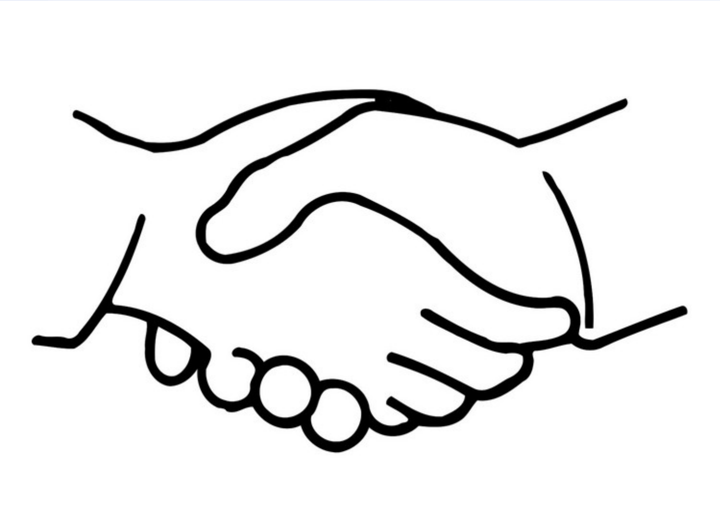
Improvements: impact

- Lower latency for event reception
  - better/faster FM radio operation
- No other outcome
  - FM SM can't handle more than one command at a time
    - Global state variables
    - Multiple race conditions
  - Async events (RDS text) may still be lost

Conclusions

- Deploying vendor software in a product...
  - Is unavoidable
  - Is to be considered carefully
- Userspace drivers
  - Provide efficient way for vendor SW redesign
    - Performance, licensing, maintenance
- Communication is very important

Q&A



Thanks for your attention!