Automotive BoF
Can Linux and Functional Safety Mix ?
(Take 2)

Robin Randhawa
ARM

Recap: My objectives

- Investigate the potential for open source software in safety critical domains
- Test assumptions with key Automotive OEMs
- Select a suitable set of open source projects – if any
- Evaluate these projects at ARM
- Promote the use of suitable projects, if any, within ARM
- Work with Linaro to try and align ARM Automotive partners around these projects

Recap: Viewpoints shared at the last Auto BoF

- What exactly is functional safety (FuSa) ?
- What are the compute trends seen in automotives over the past decades ?
- What are the primary compute partitions in a modern automotive ?
- What are the sensitivities around FuSa in those partitions ?
- Which partition is worth pursuing for Open Source System Software ?
- Why open source at all ?
- Where exactly is Linux used today in a modern automotive ?
- How do proprietary OS vendors manage to run Linux ***and*** get high safety certs ?
- What level of certification should we aim for and why ?
- What is the right system software layer to focus on ?
- What are the set of attributes this software should have ?

Desired attributes in a separation run-time*

- Open source implementation
- Small trusted computing base (in terms of LoC)
- Safety oriented architecture
- Built in security model
- Supports POSIX apps
- Supports deterministic thread scheduling
- Supports multi-core thread scheduling
- Partitioning capability using hardware assisted virtualization
- Virt machines should support multi-core guest operation
- Virt machines should support guest pass-through device access with IOMMU interop
- Inter Virt machine comms supported
- Virt machine CPU affinity expression supported

* Built from discussions with and examining offerings by Tier 1 OEMs and proprietary OS vendors

What have I been doing post Budapest ?

- Added Tier-1 OEM suppliers to my list of folk to speak to
  - Which now became Tier 1 {OEMs + **OEM suppliers** + OS Vendors}
- Tested viewpoints with them
  - Basis for this BoF
- Began appreciating the value of consolidation
  - ***All*** Tier-1 OEM suppliers I spoke to are working hard towards consolidating compute
  - That consolidation aspiration is making them wake-up to the value/potential of open source software at the separation run-time level
  - But there are push-backs (more on this later)
- Along the way: Evaluated a bunch of interesting open source implementations
  - Minix3
  - seL4
  - L4Re
- The evaluations are an ongoing exercise - happy to share views – buy me beer

Important learnings from Tier-1 OEM suppliers

- Turns out the list of desirable attributes was missing a few critical ones
- In addition to these ***technical*** attributes discussed previously:
  - Open source implementation
  - Small trusted computing base (in terms of LoC)
  - Safety oriented architecture
  - Built in security model
  - POSIX compliant C lib
  - Supports deterministic thread scheduling
  - Supports multi-core thread scheduling
  - Partitioning capability using hardware assisted virtualization
  - Virt machines should support multi-core guest operation
  - Virt machines should support guest pass-through device access
  - Inter Virt machine comms supported
  - Virt machine CPU affinity expression supported
- You also need these really hard to meet ***non-technical*** ones:
  - ***Evidence of ISO compliant development processes***
  - ***Accountability for the implementation***
  - ***Blessing of a Tier-1 OEM/OEM Supplier***
  - ***Certification friendly interfaces***

Evidence of ISO compliant development processes

- Er, Yes

Accountability for the implementation ?

- You might have the coolest open-source separation run-time with a super complete feature-matrix
- No Tier-1 OEM will use it unless there is a clearly identified entity that is responsible for the safety sign-off for that run-time
- This is probably the number 1 reason why Tier-1 OEMs shy away from open source software for higher safety integrity levels
  - “**When comes the time, who do I point the finger at ?**”

    Put more appropriately:
  - “**Who provides me with a sign-off on the safety certification and liability ?**”
- This is also why the “Big 3” proprietary vendors thrive in this space
  - QNX
  - Integrity
  - PikeOS

Blessing of a Tier-1 OEM/OEM supplier !?!

- This is sad but there is an undeniable insecurity driven Tier-1 OEM deadlock
  - Even if you have a safety certified run-time offering, open or proprietary*, with a clearly accountable entity behind it, no Tier-1 OEM will use it, unless some other Tier-1 OEM uses it first
- The Automotive industry works on complex webs of relationships built on brittle foundations of trust over decades
- You can’t suddenly appear with new~shiny and have everyone use it
- Someone needs to take the plunge first and then a cascade may happen
- This is probably the number 1 reason why the high assurance run-time space is so fragmented with many small shops pandering their wares (mostly) aimlessly

*Apart from the Big 3, of course

Certification friendly interfaces ?

- You will have lesser pain if your separation run-time is AUTOSAR compliant
- Specifically adaptive AUTOSAR
  - The spec is still baking for Adaptive AUTOSAR but jumping in early will be worthwhile
- Adaptive AUTOSAR is an OS interface for modern automotive use cases, such as:
  - Autonomous driving
  - Vehicle – to – Vehicle (V2V) comms
  - Multimedia
  - OTA updates

Is this even possible ?

- Desirable attributes in a separation run-time
  - ***Open source implementation***
  - Small trusted computing base (in terms of LoC)
  - Safety oriented architecture
  - Built in security model
  - POSIX compliant C lib
  - Supports deterministic thread scheduling
  - Supports multi-core thread scheduling
  - Partitioning capability using hardware assisted virtualization
  - Virt machines should support multi-core guest operation
  - Virt machines should support guest pass-through device access
  - Inter Virt machine comms supported
  - Virt machine CPU affinity expression supported
  - ***Proof that ISO compliant development was done***
  - ***Accountability for the implementation***
  - ***Blessing of a Tier-1 OEM/OEM Supplier***
  - ***Certification friendly interfaces***

Maybe. Enter The Cathedral and the Bazaar (or Unicorn)

- Based on everything discussed so far, I think the ideal project would be:
  - An open source offering (of course)
  - Provably mature by way of commercial use (don’t start from scratch)
  - Has a split development model: flexible open instance + rigid closed instance
  - Flexible open instance: developed as usual in the open with community participation
  - Rigid closed instance: developed by an owning entity (possibly commercial)
  - Rigid instance aligns with the open instance at a cadence dictated by necessity and certification cost
  - The owning entity has experience with assessment and certification
  - Has ideally already been down this route before
  - Has ideally gotten the blessing of a Tier-1 OEM by way of product deployment
- An open source community helps enrich the open instance at a suitable pace by open collaboration. Everyone benefits from this instance.
- The owning entity maintains the rigid instance and takes on the certification qualification overheads. Tier-1 OEMs who want assurances engage with the owning entity (they get to point the finger).
- Everyone is happy and drives into the sunset.

Interesting options: L4Re

- L4Re – the L4 Run-time Environment
  - A GPLv2 “microvisor” implementation by KernKonzept
- Appears to tick all the technical boxes
  - Written specifically for mixed criticality compositions (Apps with differing safety, security and real-time requirements)
  - Typical micro-kernel design (minimal trusted compute base – only does address spaces, threads and IPC in the kernel – everything else in user space (device drivers, apps, policy)
  - Provides native programming model for high criticality threads (POSIX compliant micro-apps) running directly under the microkernel
  - Provides a trusted hypervisor for rich/legacy OS’
  - Provides a paravirtualized Linux implementation (L4Linux) – like UML but for running Linux as an L4Re process
  - Built in security architecture
  - Built in anti resource starvation architecture
- Appears to tick most of the non-technical boxes too
- ***Come see the L4Re session***
  - **“Preventing Linux in your car from killing you: The L4Re Open Source Microvisor System”**
  - **SFO17-416: Thursday 1100 @ Session Room 2 Cypress Room B**

Interesting options: seL4

- Project claim: “The world’s first OS kernel with end to end proof of implementation correctness and security enforcement”
- GPLv2 implementation by Data61
- Uses formal methods for machine checked proof of kernel implementation correctness
- Proof that the C implementation is free from classic C problems
  - Buffer and stack overflows
  - Null pointer exceptions
  - Use-after free
- Proof that there are no compiler introduced bugs
  - Uses formal models of the ISA for a given processor architecture revision
- Ticks a lot of technical boxes (virtualization, multi-core etc)
- Does not tick a lot of non-technical boxes but could be a solid longer-term open source option
- ***Come see the seL4 session***
  - **“Using math to prevent Linux in your car from killing you: The open source seL4 kernel”**
  - **SFO17-417: Thursday 1600 @ Session Room 2 Cypress Room B**

Interesting options: Xen

- The Xen question: Why don’t we consider using Xen ?
- Rephrase: What would it take to make Xen a high assurance separation run-time ?
  - Create a snapshot of the mainline
  - Get a Xen expert to prune Xen down to an assessment friendly LoC (~30K LoC)
  - Retrospectively create a specification of what remains
  - Maintain the pruned snapshot in lock step with this specification
  - Engage with an experienced assessor to iteratively fill in the missing bits
- This will likely not be Xen anymore in the strictest sense but it’s definitely a possibility
- You still need an accountable owner etc

Interesting options: Jailhouse

- A lightweight (sub 10k LoC) partitioning hypervisor contributed by Siemens
- Appears to have the right features for mixed criticality compositions
- Focus is truly on partitioning and not on virtualization
- So no scheduler, no IO emulator etc
- More evaluation needed
- Problems:
  - Linux kernel linkage: Linux used for bootstrap and control of partitions
  - Perhaps the linkage with Linux could be abstracted away ?
  - Still needs accountability etc

Next steps: A Linaro Automotive SIG

- A Special Interest Group incubated by and run under the aegis of the Linaro TSC
- Aiming to get expert viewpoints to help chart strategy on the Automotive theme
- My personal views on areas that this group should focus on:
  - Assess the open auto “middlewares” (AGL, Genivi, OSADL) for ARM architecture optimisation potential, then scope out work and execute
  - Select and progress (ideally more than one) open source separation run-time
  - Work with an accountable entity to take a run-time through to certification with member participation (so focus on the engineering and leverage a known accountable entity for certification)
  - Build a mixed criticality composition and demonstrate on partner hardware (perhaps AGL in a Linux VM running on top of a separation run-time)
  - Actively reach out to Tier-1 OEMs at an engineering level to socialise offerings
- More news at the next Connect

End