RavenSPARK – an approach to critical, hard real-time, embedded software

Rod Chapman, Altran Praxis
What we do...

• The problem: construction of “high assurance” embedded systems.
• Typical properties:
  – Embedded – little or no RTOS.
  – Hard real-time – timing requirements that we must guarantee prior to deployment.
  – Potential for significant loss in the event of failure – of life, money, reputation etc.
  – Need to generate an “assurance case” for a regulator of some sort. e.g. safety case.
Main issues...

• How to specify, refine and verify timing properties?
• “Design for timeliness”...
• WCET analysis with modern machines, programming languages and compilers.
• Schedulability analysis for multi-core machines.
A solution...

• “RavenSPARK”

• What does that mean?

• A combination of...
  – SPARK – a high-assurance programming language.
  – The “Ravenscar” approach to building concurrent software.
What is SPARK?

• SPARK = Ada subset + contracts + static verification toolset.

• Subset achieves an unambiguous semantics, so verification can be sound and fast for all standard compilers and target machines. This is very unusual!

• Verification is based on information-flow analysis, and VC-Generation based on Hoare logic and theorem proving.
What is Ravenscar?

• A village in North Yorkshire?

• A “tasking subset” designed for predictable, hard real-time programming?

• Both...
What is Ravenscar?

• Basic programming model:
  – A fixed set of *periodic* or *aperiodic* tasks.
  – A fixed set of protected objects (POs.)
  – A particular scheduling regime.

• POs are like a Hoare Monitor (for the older people), or a “synchrononized class” (for the younger members of the audience.)
  – An encapsulation of some variable state(s) and operations that access that state, which are guaranteed to be mutually exclusive.
Scheduling Ravenscar

• Scheduling regime: “fixed priority pre-emptive with immediate priority ceiling inheritance”

• Eh? What does that mean in English?
  – Fixed-priority – all tasks have a fixed basic priority.
  – Pre-emptive – the highest priority runnable task in the system is always running.
  – Priority Ceiling – POs have a “ceiling priority” which must be higher than that of all the tasks that can access it.
  – Ceiling Inheritance – when a tasks calls a PO, its priority is “boosted” up to that of the PO’s ceiling priority.
Scheduling Ravenscar

• Simple scenario: if task X is running, then it must be the highest priority task in the system. If X calls a PO, then its priority gets boosted up to the ceiling priority of the PO. X’s priority is higher than before, so it keeps running.
  – No locks or semaphores required to do this.

• Emergent property: tasks never block to enter a PO call.

• The main point: such programs are amenable to static analysis of schedulability through the so-called “rate monotonic” family of schedulability analysis algorithms.
Implementing Ravenscar

• Second point: implementing this scheduling scheme is very simple on a mono-processor – no locks at all and achievable with little or no RTOS. Minimal Ravenscar program is about 2 kilobytes object code.

• Implementations have been available for about 10 years and are suitable for certification up to the highest levels of assurance – e.g. “Level A” for commercial avionics.
Challenges

• Make it work on multi-core. Works just fine if you are willing to sacrifice schedulability analysis. We are using this approach on a big project right now.
  – We upgraded from 2-core to 8-core target machines.
  – It just worked...

• Functional verification. We need an extended Hoare logic for POs, invariants, and so on.