Multi Sloth: An Efficient Multi-Core RTOS using Hardware-Based Scheduling

Rainer Müller, Daniel Danner,
Wolfgang Schröder-Preikschat, Daniel Lohmann

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

July 10, 2014

Sloth: Let the Hardware do the Work!

Sloth kernels use hardware for OS purposes, and

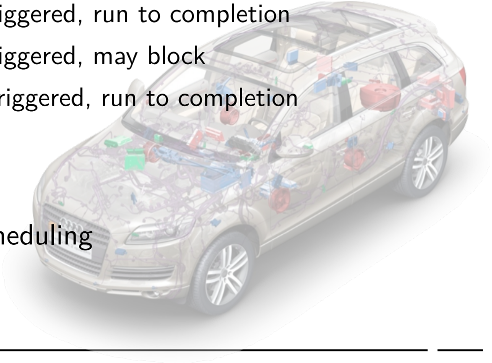
- are concise (200–500 LoC)
- are small (300–900 bytes)
- are fast (latency speed-up 2x to 170x)
- implement industry standards (OSEK, AUTOSAR OS)

Automotive Domain: The AUTOSAR OS Family

- Families of **completely statically** configured RTOS
  - **OSEK OS / OSEKtime**
  - **AUTOSAR OS**

- System model with different **control-flow types**:
  - **Basic Tasks** software-triggered, run to completion
  - **Extended Tasks** software-triggered, may block
  - **ISRs** hardware-triggered, run to completion
  - **...**

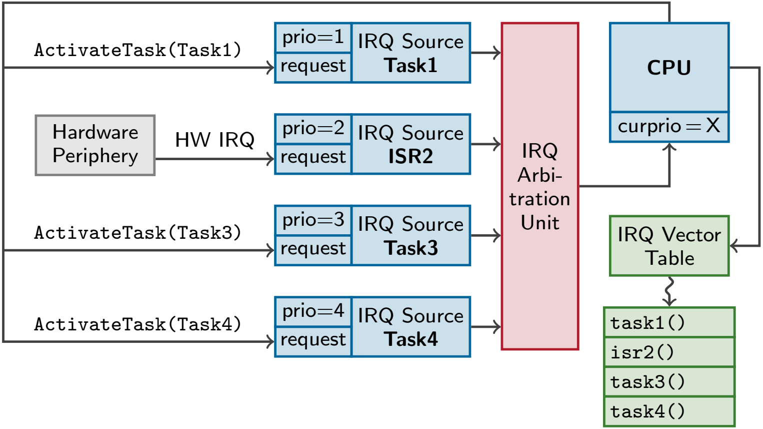
- Preemptive, **fixed-priority** scheduling



Sloth: Building Blocks

Main Idea

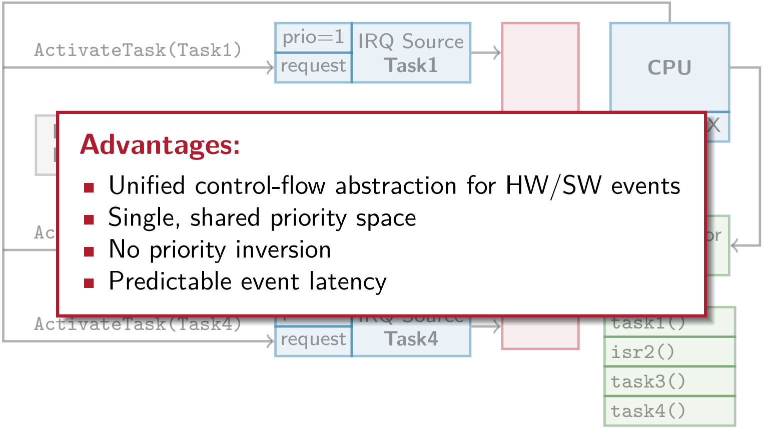
Threads are interrupt handlers, synchronous thread activation is IRQ
⇒ Interrupt subsystem does scheduling and dispatching work



Sloth: Building Blocks

Main Idea

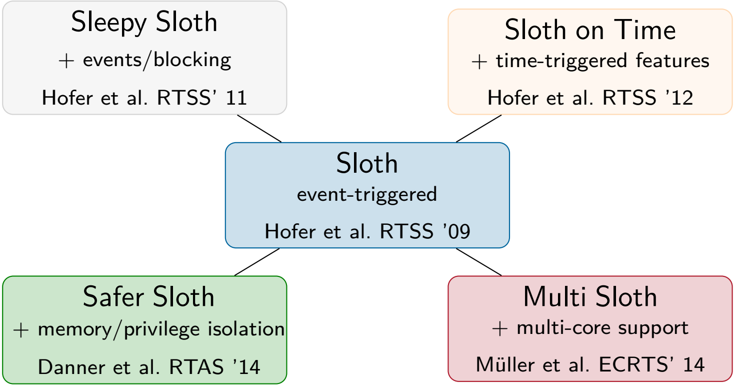
Threads are interrupt handlers, synchronous thread activation is IRQ
⇒ Interrupt subsystem does scheduling and dispatching work



Advantages:

- Unified control-flow abstraction for HW/SW events
- Single, shared priority space
- No priority inversion
- Predictable event latency

Sloth OS family



Sleepy Sloth

+ events/blocking

Hofer et al. RTSS' 11

Sloth on Time

+ time-triggered features

Hofer et al. RTSS '12

Sloth

event-triggered

Hofer et al. RTSS '09

Safer Sloth

+ memory/privilege isolation

Danner et al. RTAS '14

Multi Sloth

+ multi-core support

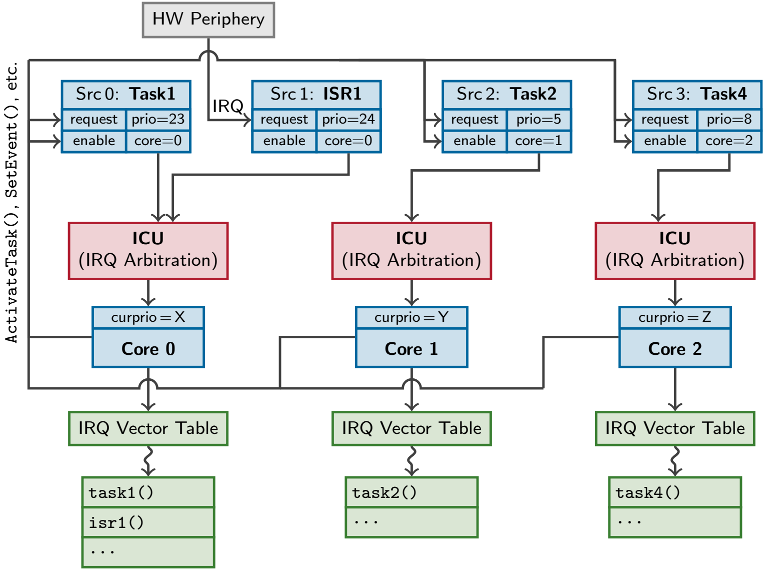
Müller et al. ECRTS' 14

Multi Sloth: Design

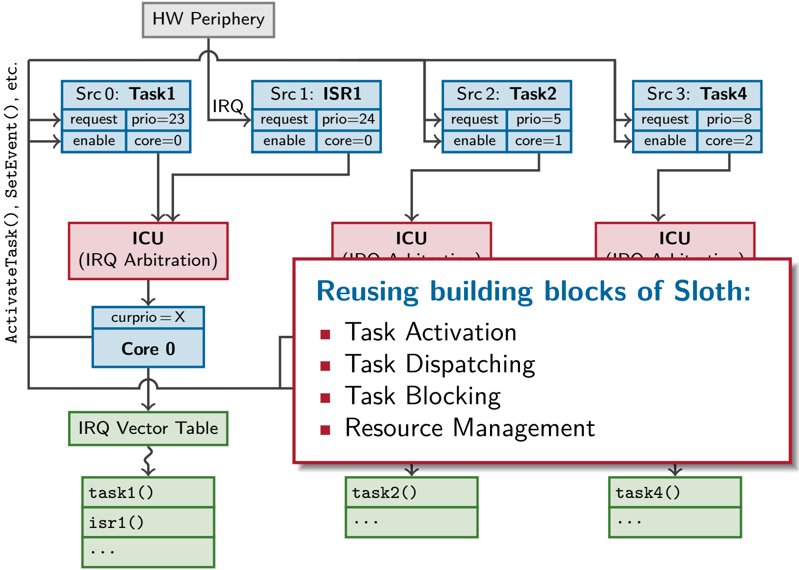
Multi Sloth
+ multi-core support
Müller et al. ECRTS' 14

- AUTOSAR system model: partitioned fixed-priority scheduling
- Reusing building blocks of Sloth
- Generative approach:
  tailoring the kernel to the application and the hardware platform
- Reference implementation for the Infineon AURIX
  - 32-bit RISC μ-Controller
  - upcoming multi-core platform in the automotive industry
  - 3 integrated TriCore CPUs
  - IRQ system with 256 priority levels and up to 1024 remappable sources

Multi Sloth: Herding Sloths



Multi Sloth: Herding Sloths



Reusing building blocks of Sloth:

- Task Activation
- Task Dispatching
- Task Blocking
- Resource Management

Synchronization in AUTOSAR OS

- Synchronization primitives as specified by AUTOSAR OS

- Local access on each core

⇒ Resources with OSEK priority-ceiling protocol
  - `GetResource()`
  - `ReleaseResource()`

- Global access across multiple cores

⇒ Spinlocks
  - `GetSpinlock()`
  - `ReleaseSpinlock()`
  - `TryToGetSpinlock()`

MPCP: Resource access synchronization

- AUTOSAR lacks definiton of semantics for spinlocks
- Requirement: priority-aware synchronization

⇒ Synchronization protocols

- Multi-processor Priority Ceiling Protocol (MPCP)
  - published by Rajkumar et al. (RTSS '88, ICDCS '90)
  - proposed for AUTOSAR OS by Lakshmanan et al. (SAE 2011)
- Resuing building blocks to implement MPCP in Multi Sloth

MPCP for AUTOSAR OS System Model

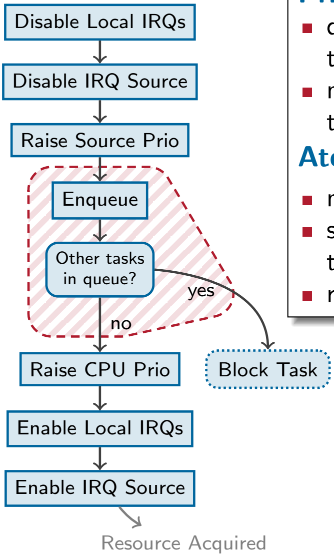
- All tasks use their assigned priority except in critical sections, all cores share the same priority space
- Local resource access is synchronized using OSEK single-core PCP
- Each global resource has a ceiling priority above all task priorities

$$\pi_{res} = \pi_{max} + \pi_{task}$$

- Acquiring a global resource **raises the current execution priority** to the ceiling priority
- If resource is already held by another task, the requesting task **blocks**
- When a task leaves a global critical section, the highest-priority task waiting for this global resource is signaled and **resumes at the higher priority**

MPCP: Implementation for Infineon AURIX

`GetMPCPResource():`



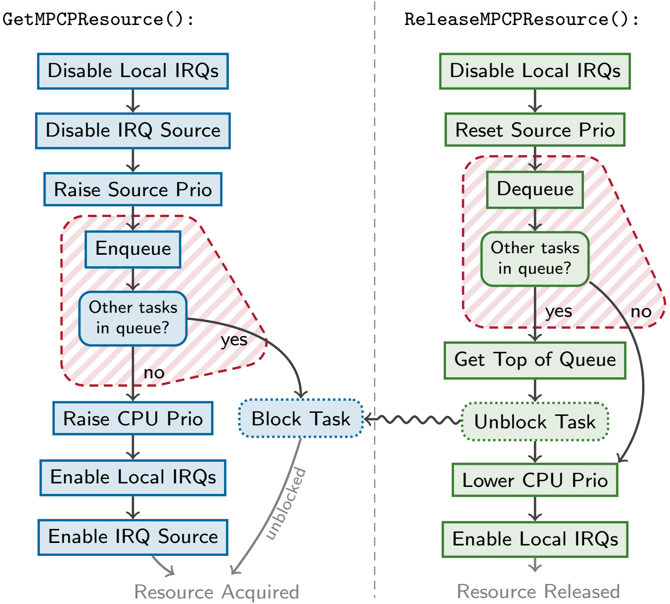
Priority-ordered waiting queue

- queue as bitmask, one bit for each task accessing this resource
- most significant bit for highest-priority task marks task holding the lock

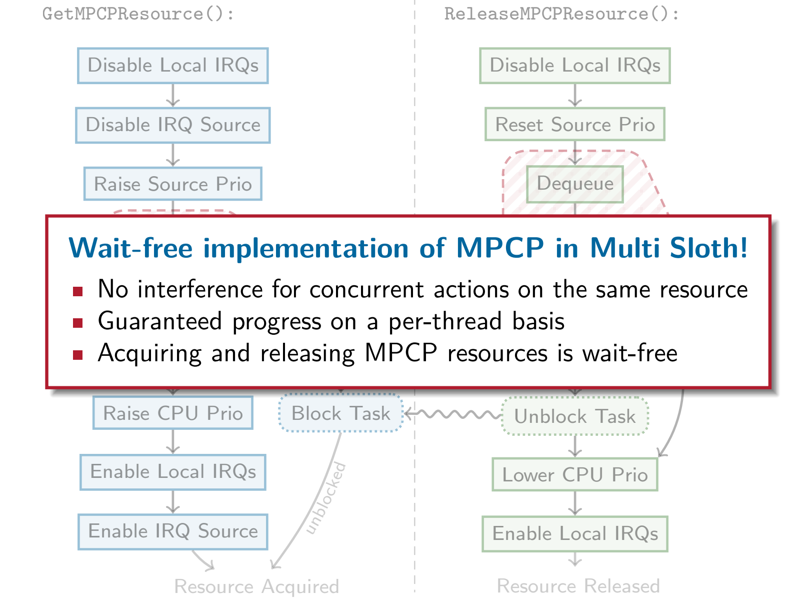
Atomic enqueue with `swapmsk`

- new instruction on Infineon AURIX
- swaps single bits in a word according to a bit mask
- returns previous data word

MPCP: Implementation for Infineon AURIX



MPCP: Implementation for Infineon AURIX



Wait-free implementation of MPCP in Multi Sloth!

- No interference for concurrent actions on the same resource
- Guaranteed progress on a per-thread basis
- Acquiring and releasing MPCP resources is wait-free

Evaluation

- Microbenchmarks of system service overheads
  - **Basic** system on the Infineon AURIX

Transition	Cycles
`ActivateTask` w/o dispatch	65
`ActivateTask` w/ dispatch	87
`ChainTask` w/ dispatch	97
`GetResource`	36
`ReleaseResource` w/o dispatch	19
`ReleaseResource` w/ dispatch	41
`TerminateTask` w/ dispatch	20
`ActivateTask()` cross-core round-trip	135

Evaluation

- Microbenchmarks of system service overheads
  - **MPCP** resources on the Infineon AURIX

Transition	Cycles
`GetMPCPResource()` w/ blocking	217
`GetMPCPResource()` w/o blocking	112
`ReleaseMPCPResource()` w/ local dispatch	360
`ReleaseMPCPResource()` w/o dispatch	134
`ReleaseMPCPResource()` w/ remote unblock	183
`ReleaseMPCPResource()` w/ local unblock and dispatch	311
`ReleaseMPCPResource()` w/o unblock w/ dispatch	231

Conclusion

- Multi Sloth ...
  - is an efficient multi-core AUTOSAR OS
  - implements MPCP: multi-core systems beyond AUTOSAR
  - offers low latency with predictable overheads
  - adopts design philosophy of the Sloth kernel family