SLOTH: Threads as Interrupts

Wanja Hofer, Daniel Lohmann, Fabian Scheler,
Wolfgang Schröder-Preikschat

Friedrich-Alexander-Universität
Erlangen-Nürnberg

30th IEEE Real-Time Systems Symposium
December 3, 2009

Rate-Monotonic Priority Inversion

- Problem: high-priority threads disturbed by low-priority ISRs
- Solution 1 [CASES '09]: co-processor pre-handles IRQs, interrupting the CPU only when necessary
- Solution 2 [RTSS '09]: threads in the same priority space as ISRs; i.e., threads as interrupts

SLOTH: Threads as Interrupts

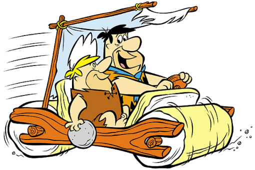
- **Idea: threads are interrupt handlers, synchronous thread activation is IRQ**

- Let interrupt subsystem do the scheduling and dispatching work
- Applicable to priority-based real-time systems
- Advantage: small, fast kernel with unified control-flow abstraction

Talk Outline

1. OSEK terminology and scope
2. SLOTH design
3. Design implications
4. Performance evaluation
5. Limitations

OSEK in 90 Seconds

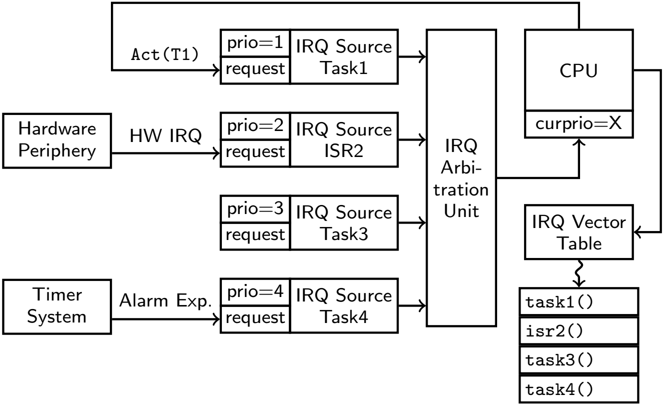


- Standard developed by automotive industry
- Statically configured, event-triggered OS

- **Tasks** scheduled and dispatched by OS scheduler
- **Category-2 ISRs** can call system services, need kernel sync
- **Category-1 ISRs** cannot call system services, do not need sync

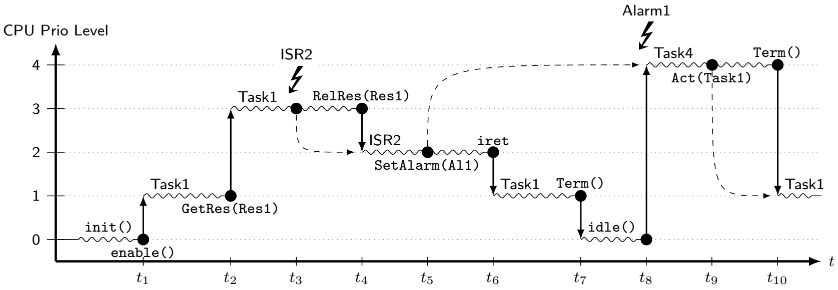
- **Resources** for application sync, with stack-based priority-ceiling protocol
- **Alarms** configured to activate task or execute callback upon expiry

Sloth Design



- Platform must support IR priorities and software IR triggering

Example Control Flow



Design Implications

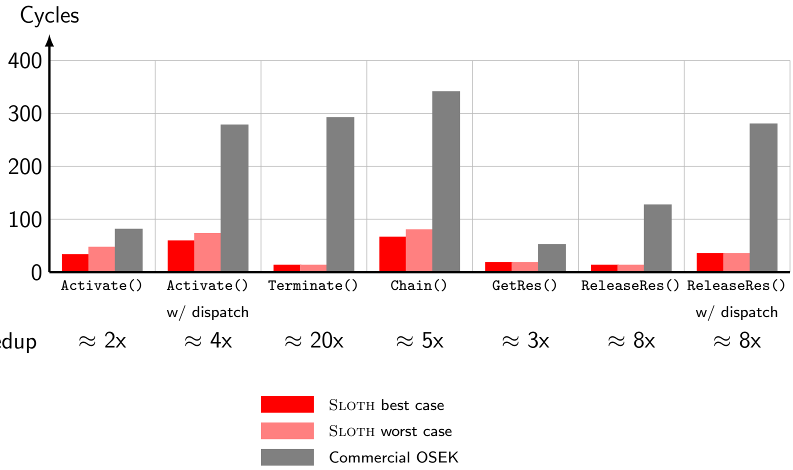
- Concise kernel design and implementation ($<$ 200 LoC, $<$ 700 bytes)
- Single control-flow abstraction for tasks, ISRs (1/2), callbacks
  - Handling oblivious to how it was triggered (by hardware or software)
- Unified priority space for tasks and ISRs, no rate-monotonic priority inversion
- Straight-forward synchronization by altering CPU priority
  - Resources with ceiling priority (also for ISRs!)
  - Non-preemptive sections with `RES_SCHEDULER` (highest task priority)
  - Kernel synchronization with highest task/cat.-2-ISR priority

Performance Evaluation: Methodology

- Implementation on Infineon TriCore (widely used in automotive)
- Evaluation of task-related system calls:
  - Task activation
  - Task termination
  - Task acquiring/releasing resource
- Performance of other system calls and application similar to traditional systems
- Comparison with performance of commercial OSEK implementation

- Two numbers for SLOTH: best case, worst case
- Depending on number of tasks and system frequency

Performance Evaluation: Results



Speedup	≈ 2x	≈ 4x	≈ 20x	≈ 5x	≈ 3x	≈ 8x	≈ 8x

Limitations of the SLOTH Approach

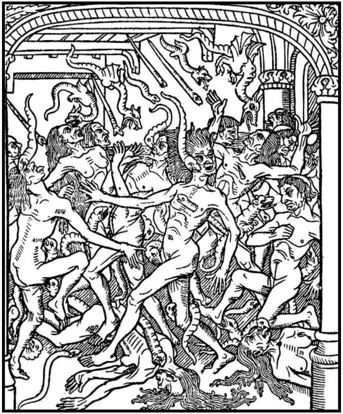
- No blocking tasks (OSEK: extended tasks with events): not compatible with stack-based execution
- No multiple tasks per priority: execution order has to be the same as activation order

Summary

- Sloth implements tasks by using IRQs and interrupt handlers on commodity hardware platforms
  - Makes tasks a low-overhead abstraction
  - Avoids rate-monotonic priority inversion
  - Keeps software footprint low
- Sloth implements the BCC1 class of OSEK

SLOTH: One of the Seven Deadly Sins

David Fincher, Se7en (1995)



Nicolas le Rouge, Le Grant Kalendrier Des Bergiers (1496)